Title page:

Text: An R-package for Analyzing and Visualizing Human Language Using Natural Language Processing and Deep Learning

Oscar Kjell[12*], Salvatore Giorgi[3] & H. Andrew Schwartz[2]

[1] Lund University, Department of Psychology
[2] Stony Brook University, Department of Computer Science
[3] University of Pennsylvania, Department of Computer and Information Science

*corresponding author; oscar.kjell@psy.lu.se

Note. Kjell was funded by the Swedish Research Council (2019-06305); Schwartz was funded by a National Institutes of Health-NIAAA grant (R01 AA028032). Tutorial data, models and code: https://osf.io/dgczt/.

Abstract

The language that individuals use for expressing themselves contains rich psychological information. Recent significant advances in Natural Language Processing (NLP) and Deep Learning (DL), namely *transformers,* have resulted in large performance gains in tasks related to understanding natural language such as machine translation. However, these state-of-the-art methods have not yet been made easily accessible for psychology researchers, nor designed to be optimal for human-level analyses. This tutorial introduces *text* (www.r-text.org), a new R-package for analyzing and visualizing human language using *transformers,* the latest techniques from NLP and DL. The *text package* is both a *modular solution* for accessing state-of-the-art language models and an *end-to-end solution* catered for human-level analyses. Hence, *text* provides user-friendly functions tailored to test hypotheses in social sciences for both relatively small and large datasets. This tutorial describes useful methods for analyzing text, providing functions with reliable defaults that can be used off-the-shelf as well as providing a framework for the advanced users to build on for novel techniques and analysis pipelines. The reader learns about three core methods: 1) *textEmbed*: to transform text to traditional or modern transformer-based word embeddings (i.e., numeric representations of words); 2) *textTrain* and *textPredict*: to train predictive models with word embeddings as input, and use the models to predict from; 3) *textSimilarity*: to compute semantic similarity scores between texts. The reader also learns about two extended methods: 1) *textSimilarityTest*: to significance test the difference in meaning between two sets of texts; and 2) *textProjection* and *textProjectionPlot*: to examine and visualize text within the embedding space according to latent or specified construct dimensions (e.g., low to high rating scale scores).

Translational abstract

Natural language is the fundamental way individuals communicate their thoughts and emotions to others. Recent advances in Artificial Intelligence (AI), referred to as *transformers,* have resulted in large increases in performance at most tasks related to understanding natural language. This tutorial introduces how to use these state-of-the-art AI techniques in both custom research analyses as well as in completely end-to-end analytic processes. We describe *text*, a software package which provides transformer-based techniques intended to be easily accessible for social scientists. *Text* is *open-source*, written for the statistical programming language *R*, and it is free to use or alter. It comprises user-friendly functions to transform text to numeric representations,

that, for example, are used for examining their relationship to other variables or for visualizing statistically significant features of texts. Transformers can facilitate analyses of natural language for gaining psychological insights with unprecedented accuracy and thus provide a more detailed understanding of the human condition.

Tutorial data, models and code: https://osf.io/dgczt/.

Introduction

How individuals express themselves and their state of mind with natural language constitutes a wealth of information for understanding them psychologically and socially (e.g., see Kern et al., 2016; Kjell et al., 2019). “Language is the most common and reliable way for people to translate their internal thoughts and emotions into a form that others can understand.” (Tausczik & Pennebaker, 2010, p. 25). This tutorial explains how psychology researchers can use recent advances in Artificial Intelligence (AI), including Natural Language Processing (NLP) and Deep Learning, to quantitatively analyze natural language.

Researchers in NLP have turned to open-vocabulary methods that rely on patterns in linguistic data to derive models of language. These methods leverage the idea that words may be represented by values based on how they co-occur in languages (e.g., Firth, 1957), and allow for modeling words according to the contexts in which they appear, rather than relying on a priori assumptions about word-category relations resulting in extremely powerful models of language. In fact, the latest version of such models, the deep learning-based *transformers,* have led to nothing short of a transformation in the AI field concerned with language: natural language processing. Researchers have called for the use of these language models to gain psychological insights (e.g., Eichstaedt et al., 2020).

The transformers method has universally increased the accuracy of AI techniques for processing language. For example, the transformer-based model GPT-3, by OpenAI, is writing documents believed to be authored by humans (Brown et al., 2020); and BERT (Bidirectional Encoder Representations from Transformers; Devlin et al., 2019), the first widely used, general purpose transformer network developed by Google, has already been referenced in over 16,000 scholarly works in a period of only 2 years[1].

The key to transformers' success is that such models are able to represent words differently according to the context they are in. BERT was released by Google and has been integrated into its Search function; now it actually understands the difference between “travel from Sweden to New York” and “travel from New York to Sweden” (Nayak, 2019). This tutorial aims to make these methods easily available to a broad audience within social and behavioral sciences; as well as further developing and optimizing them for human-level analyses. These state-of-the-art word embeddings may be used to examine their relationships to (i.e., predict) numerical variables, compute semantic similarity to other texts, statistically test the difference in meaning between

other texts, or visualize (statistically significant) words in various dimensions within the word embedding space.

For this discussion, the type of data may broadly be categorized into two (overlapping) types: First, *everyday occurring language*, which, for example, include gaining insights from analyzing recordings of spoken language in relation to emotional fluctuations throughout the day (Sun et al., 2020); and social media text (Park et al., 2014) to predict both physical (Eichstaedt et al., 2015)

---

[1] Google Scholar accessed March 11, 2021

and psychological (Eichstaedt et al., 2018) outcomes. Examining naturally occuring language may also include analyzing emails, letters, blogs, text messages, medical journals, speeches, diaries, song texts, voice recordings etc., though one must consider ethical and privacy issues when analyzing such text.

Second, probed *language* involves probing or asking participants to answer questions with spoken or written language. These types of data have, for example, been used to measure and describe psychological constructs through probed language-based assessments as a complement to traditional rating scales (Kjell et al., 2019; Kjell et al., 2022), and analyze written narratives of traumatic life events to predict health related outcomes (Campbell & Pennebaker, 2003; Son et al., 2020). Probled language may also involve asking participants to recall various memories, describe themselves in various ways, partake in stream-of-consciousness tasks and so on.

In addition, language may be generated within an experimental context. Computational language methods can, for example, be used to enhance experimental control by matching word stimuli according to semantic similarity (Dougal & Rotello, 2007; Gagné et al., 2005), examining the text generated from experimental manipulations (Garcia & Sikström, 2013) and using semantic similarity of the names of objects to find significant correlations with neural organizations in the brain (Carlson et al., 2014). The multitude of applications of computational methods highlights its potential and flexibility as a research method.

This tutorial covers methods in an R-package called *text* (www.r-text.org) to carry out the use of transformers for psychological research. *Text* is an open-source library comprising tools for analyzing and visualizing various features of texts; both in relation to other texts as well as numerical variables. The package makes state-of-the-art natural language processing (NLP), statistics, and machine learning (ML) techniques available to R users, with functions particularly targeting social scientific applications. To map words to numeric representations, *text*-functions transform texts to word embeddings using transformer-based pre-trained language models. This tutorial assumes very basic knowledge of R (R Core Team, 2022); so, for a free, beginners tutorial see *R for Data Science* by (Grolemund & Wickham, 2018).

Objectives and Aims

The *text*-package incorporates two main objectives. First, to serve R-users as a *point solution* for converting text to contextual word embeddings – numeric representations of words – using a

state-of-the-art AI technique called *transformers*. These word embeddings may be used for a large variety of tasks in the user's own analyses (pipeline). The second objective is to serve as an *end-to-end solution, where text* provides powerful and accessible functions for analyzing and visualizing text in relation to other text and numerical variables. This tutorial describes both core and extended functionalities. Core functionalities are based on standard methods with empirical support, including: 1) transforming text to word embeddings, 2) predictive modeling with word embeddings, and 3) computing semantic similarity scores between text. The extended functionalities are less well-established, novel methods, aimed at supporting the core

functionalities, including 1) significance testing the difference between two sets of texts based on semantic similarity scores, and 2) visualizing words' position in the embedding space. . The objective of the *text*-package is to balance user-friendliness and flexibility whilst simultaneously empowering with advanced analyses. Accessibility is reflected in functions with reliable default settings selected by experts in the NLP fields so that those with no experience can use out of the box settings to test their research hypotheses. Further, visualizations aim to make it easy to understand what is going on in the analyses; for example, displaying actual word embeddings, plotting words, or viewing cross-validated predictive results.

Current Alternatives

There are few alternatives in R (R Core Team, 2022) that focus on getting state-of-the-art word embeddings; and that have functionalities tailored for analyzing embeddings in downstream tasks relevant for social sciences and psychological research. Computer scientists have predominantly used Python (Van Rossum & Drake Jr, 1995), and Python-libraries such as the Differential Language Analysis ToolKit (DLATK; Schwartz et al., 2017), PyTorch (Paszke et al., 2019), spaCy (Honnibal, & Montani, 2017), and NLTK (Bird, Klein, & Loper, 2009). Further, even though computer scientists use Python for standard NLP/ML tasks, few python packages attempt to bridge this line of work with psychological research such as DLATK. *Text* comprises an interface with Python to get the state-of-the-art language models whilst also conceptually building on DLATK, and drawing on experiences learned from that project (here also see the web-application Semantic Excel; Sikström et al., 2018).

There are some R-packages that allow for text mining (e.g., see tm [Feinerer & Hornik, 2019], tidytext [Silge & Robinson, 2016], and quanteda [Benoit et al., 2018]) and for various automatic coding procedures of text responses (see shinyReCoR; Andersen & Zehner, 2021). To our knowledge, there is only one other R-package that enables the transformation of text to word embeddings (RBERT; an R implementation of BERT; Bratt & Harmon, 2020), however, it does not come with functions to use these word embeddings including ML techniques. Additionally, *text* comprises functions tailored for small data as well as big data. Whereas other R-packages focus on text mining of big data, *text* enables transformer-based embeddings as well as analyses of relatively small datasets, which is often common in social and behavioral sciences.

Installation of text

The *text*-package can be downloaded and installed from CRAN or Github. *Text* uses an R-package called *reticulate* (Ushey et al., 2020) as an interface to Python (Van Rossum & Drake Jr, 1995) and the Python packages *torch* (Paszke et al., 2019)*, transformers* (Wolf et al., 2019), nltk (Bird, Klein, & Loper, 2009) and numpy (Oliphant, 2006). It is important to get versions of these libraries to work together. The text-package includes functions that help you set up an environment with the correct versions of these packages for your specific OS (which is tested for Linux, MacOS and Windows).

Tutorial data, models and code can be retrieved at the open science framework at: https://osf.io/dgczt/. To install the text-package run:

```
# Install text from CRAN
install.packages("text")
library(text)

# Set-up en environment with text-required python packages
textrpp_install()

# Initialize the environment - and save the settings for next time
textrpp_initialize(save_profile = TRUE)
```

If you experience problems with the installation see the extended installation guide for up to date information (https://r-text.org/articles/Extended_Installation_Guide.html).

Core Functionality

The *text-package* has three core functionalities: (1) transforming text to word embeddings, (2) predictive modeling with word embeddings, and (3) comparing words or texts for semantic similarity. These are described below, followed by the introduction of three extended functionalities.

Transforming text to state-of-the-art word embeddings

NLP is concerned with automatically processing and making sense of digitized human language (for a broad and detailed treatment of NLP methods see Jurafsky & Martin, 2020). The idea that each word in human languages may be represented by numeric vectors dates back at least to the 1950s (e.g., see Firth, 1957; Osgood et al., 1957). Despite its long history, the field of NLP has undergone a sort of revolution over the past three years, seeing substantial advances in the state-of-the-art. These advances span over nearly every common task that NLP attempts to solve including syntactic parsing, sentiment analysis, question answers, and machine translation. All advances are tied to a single method: *Transformers* (Devlin et al., 2019; Rogers et al., 2020;

Vaswani et al., 2017).

The *text*-package implements these advances to enable the user to access many state-of-the-art pretrained language models. In this next part, word embeddings are described in more detail, followed by a demonstration on how the *text*-package transforms text data into word embeddings. These word embeddings are later used in down-stream tasks to provide examples of how they may be used, including predicting psychology related outcomes (e.g., psychological rating scale scores), examining similarity in meaning between texts, and testing whether two sets of texts

statistically differ in meaning. Hence, word embeddings may be seen as the backbone of most *text*-functions.

Word Embeddings

A word embedding is a list of values (an ordered *vector*) that aim to numerically represent the meaning of a word. A word embedding normally comprises several hundred numbers; so that a word is represented by many dimensions. The numbers may be seen as coordinates in a geometric space that comprises several hundred dimensions (a high dimensional space). The closer two words are in this space (i.e., the more similar their *vector* embeddings are), the more similar the words are expected to be in meaning. In other words, embeddings capture the relationships between words, where proximity in the high dimensional embedding space signifies similarity in meaning. To represent several words, sentences and paragraphs, word embeddings may be aggregated. The aggregation can, for example, be through the mean, maximum or minimum of each dimension of the word embedding.

To train word embeddings with high quality requires a lot of text data; where the approaches use the statistical patterns of *how* words are used. “You shall know a word by the company it keeps” (Firth, 1957, *p.* 11) is a core rationale underlying the creation of word embeddings. Words within natural language are *not* randomly distributed, instead contextual words are predictable and define its meaning (Iliev et al., 2014). Hence, it is possible to use this distribution to represent “the meaning of a word through the contexts in which it has been observed in a corpus” (Erk, 2012*, p.* 635). This may be achieved by constructing a table of word co-occurrence counts, where the dimensions are extracted using a dimension reduction technique such as the Singular Value Decomposition (Golub & Kahan, 1965). Practically, the first column of the frequency table may hold the words in a natural language, the first row may hold word contexts (such as documents of texts) and the rest of the cells hold the co-occurrence counts/frequency. Ultimately, the words are represented by a vector containing a number for each extracted dimension. Note that these types of approaches are referred to as *unsupervised,* as no predefined categories or judges are used to produce the numeric representations.

Decontextualized word embeddings: A bag of words

Decontextualized word embeddings (or *semantic representations* as they were called) do not account for the context a word was in. Earlier methods were unable to capture the order that words appeared in, so that words in a text were treated as a bag of scrambled words: Bag of words (BOW) models (see Figure 1). The word embedding is thus *static* (or decontextualized), so

that the embedding for “happy” is always the same even if it appeared in the context of “I am happy” versus “I am not happy”. Examples of commonly used decontextualized approaches include word2vec (Mikolov et al., 2013), Latent Semantic Analysis (LSA; Deerwester et al., 1990) and Latent Dirichlet Allocation (LDA; (Blei et al., 2003).

Contextualized word embeddings and word order

Contemporary NLP algorithms aim to extract the latent meanings in text. These algorithms are based on deep neural net (or deep learning) architectures to construct contextualized word

embeddings, using decontextualized word embeddings as input (e.g., see BERT; Devlin et al., 2019). When using these algorithms, a word's embedding is different depending on the context it was in and the order of the words in the context – this is achieved by enabling the word embedding to be influenced by other word embeddings in the context through a mechanism referred to as self-attention.

The Deep Neural Network Architecture: The Layers and Hidden states

As a deep learning transformer model, BERT comprises several layers. *Hidden states* refer to the output of each layer; these may be aggregated across layers to a word embedding that represents each word (or sentence). The use of multiple layers enables the models to capture nonlinear relationships. BERT-base comprises 12 layers, whereas BERT-large comprises 24 layers; where each layer comprises numeric values for each dimension, which for BERT is 768 dimensions. Hence, with BERT-large one token can be represented with 18,432 (24 x 768) values. There are several ways to make use of these layers: including only using one of the layers, or concatenating them to one longer embedding.

It is not yet fully understood how the various layers differ; and the advice on how to best use them differs. In short, empirical examinations indicate that "BERT's intermediate layers encode a rich hierarchy of linguistic information, with surface features at the bottom, syntactic features in the middle and semantic features at the top." (Jawahar et al., 2019, *p*. 1). Further, it has been demonstrated that later BERT layers are more context-specific (Ethayarajh, 2019). Below we discuss how to use these layers in human level tasks.

Self-Attention

Transformer's deep neural net architecture has the *self-attention* mechanism at its core (Vaswani et al., 2017). It is self-attention that enables the model to combine information about surrounding words at varying degrees. Thus, each surrounding word can influence the word embedding of the target word. In other words, self-attention enables the algorithm to incorporate the embeddings of other words' embeddings in so far as it produces a more robust representation of the target word. The self-attention mechanism puts *different* attention or weight to different word embeddings from the context depending on which appear to be most relevant. During training, the transformer-based models have learned how to parameterize the attention layers in order to amplify the influence of the most relevant parts of a context.

The transformers' self-attention mechanism may be compared with previous models such as Long Short-Term Memory models (LSTM; Hochreiter & Schmidhuber, 1997). These models are based

on recurrent neural networks, where the influence of the context mechanism is less pervasive than in transformers. In LSTM the influence of the context is only based on the previous *state* – the embedding of the word just to the left of the target word, which in turn was influenced by its surrounding words, and so on. The model can also be run across the sequence backwards, such that the previous *state* is an embedding of the word to the right. But it still requires for non-adjacent pairs of words to pass information about each other through the embeddings between them rather than directly. With self-attention *all* words are connected with the potential to influence each other's meaning through the word embeddings. The transformers can model the dependency of

every two words in a text sequence, and thus are also suitable for long range dependencies. The architecture of the transformers also reduces the amount of sequential computational steps as compared to previous contemporary models such as LSTMs; and this results in decreased information loss and faster training times through parallel processing (Wolf et al., 2019).

These descriptions are at a very high level aimed to provide the reader with a sufficient understanding to be able to make use of the key functionality of transformers: turning text into feature vectors (embeddings). There are many recent papers specifically introducing deep learning with a focus on natural language processing (e.g., see Lauriola et al., 2022), in length discussing the opportunities and risks of using these models (Bommasani et al., 2021) as well as reviews/surveys providing a broader overview and deeper understanding about various model architectures of neural networks such as recurrent, recursive, convolutional, and attention models (Babić et al., 2020; Otter et al., 2019; Qiu et al., 2020), and specifically focusing on the BERT models (Rogers et al., 2020).

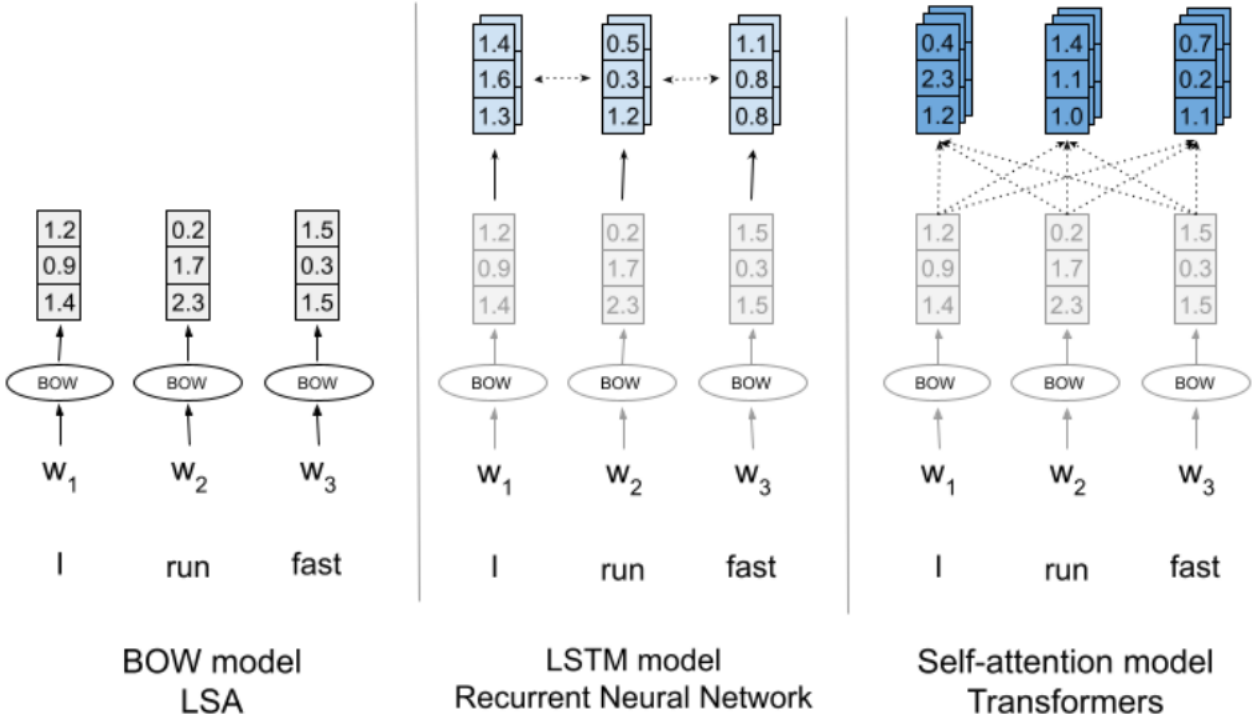


Figure 1. An illustration of how information is connected within the output of different language model architectures.
BOW = bag of words; LSA = Latent Semantic Analysis; LSTM = Long Short-Term Memory; w = word; squared symbols = word embeddings; gray = contextualized word embeddings; stronger shades of blue indicate more contextualized word embeddings.

Performance of BERT

Devlin et al. (2019) demonstrated that BERT obtains substantial improvement on a wide range of NLP tasks. For example, it achieved a 7.7%-point absolute improvement on the language understanding tasks called GLUE (General Language Understanding Evaluation); and a 4.6% absolute improvement accuracy on the MultiNLI (the Multi-Genre Natural Language Inference corpus, which comprises several hundred thousands of sentence pairs).

Accessible Pretrained Language Models in *text*

To get contextualized word embeddings for your text data, the *text*-package connects with HuggingFace's *Transformers* library (Wolf et al., 2019) in Python. This enables the users to implement many state-of-the-art *pre-trained* language models, including XLnet (Yang et al., 2019), RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2019). As the name suggests, a pre-trained model has already been trained on other data; so that the model can be applied to your text to retrieve high-quality embeddings. This is good considering that training a high-quality deep learning language model requires a lot of computational resources; Wolf et al. (2019) point out that RoBERTa was trained on 160 GB text, and that training this on a typical cloud computing service would cost around 100K USD. Consequently, some models have been developed with the focus of being smaller and requiring less computational resources to be trained (e.g., see DestilBERT; Sanh et al., 2019), whereas others have had the focus on achieving improved performance (e.g., XLNet; Yang et al., 2019).

Some models include multiple languages such as multilingual BERT (mBERT), which is trained on text from the top 104 languages with the largest Wikipedia entries. Hence, the same model includes several languages, where similar languages (e.g., Germanic, Slavic) are found close to each other in the embedding space (Libovickỳ et al., 2019). mBERT is found to learn cross-lingual word alignment with high-quality (Libovickỳ et al., 2019). *Text* implements these multilingual models.

Model and Word Embedding Specifics

Different models may be based on different types of (domain) text, and use different tokenization (described in detail below), number of layers and hidden states. It is useful to consider these aspects when selecting an appropriate model and its settings, although default settings in *text* will often achieve apt results.

Type of domain text used in pre-training

Word embeddings become better when there is high domain similarity between the text used to create the language model and the text that should be interpreted/used for down-stream tasks. To create models for a narrow domain can however be challenging because pretrained models require large quantities of text data (and computational resources). Nevertheless, it is worth thinking about the type of text used to train the models, and how that might influence your results. BERT was originally trained using Wikipedia text; however, other BERT models have been fine-tuned on clinical text (Alsentzer et al., 2019) and scientific text data (Beltagy et al., 2019). Unfortunately, these models are not available through HuggingFace (yet).

Uncased versus Cased models

Uncased models convert all words to lowercase, whereas *cased* models keep the casing of the text. Select a *cased* model if you think that letter casing will be helpful for your analyses. If you do not think that casing will be important for your task, it is likely better to select an *uncased* model since it comprises fewer words (e.g., happy and Happy are not two different words); and thus has had the chance to learn more high-quality embeddings.

Sequence length of the input

You can only impute a limited amount of text (tokens) to many language models at the time. For the BERT model the standard limit is 512 tokens at a time. This means that each word can be contextually influenced by a maximum of 511 tokens. If longer sequences of text are submitted to the *text-package* function, the text will be split up in smaller chunks; and then the word embeddings may be aggregated to represent the entire text.

Tokens and Tokenizers

A token is a string of characters with a meaning; and a tokenizer is a function that splits up sentences and words to tokens. Tokenizers may differ across models. Each token gets its own embedding. Common words are tokenized as themselves, whereas uncommon words that are not part of a pretrained model will be tokenized into smaller parts each having their own embedding. Punctuation characters (e.g., ".", "!" and "?") also tend to get tokenized as individual tokens. BERT also uses tokens to indicate delimiter to each input: [CLS] and [SEP]. The [CLS] token is always put to indicate the start of the sequence and it is often used to represent the whole sequence. The [SEP] token stands for *separator*, and this tag is used to separate sentences during the pre-training tasks of BERT which include predicting the next sentence.

As an example, the sentence: "I'm feeling relatedness with others" is tokenized to "[CLS] I ' m feeling related ##ness with others [SEP]"; where "##" indicates that "ness" was originally attached to "related". "Relatedness" was tokenized this way because it was not present within (this version of) BERT. Instead, the two word embeddings for "related" and "ness" are aggregated to represent relatedness.

Layers

To select the type of model and number of layers may be based on systematic, empirical research. Two systemic studies evaluating different transformers models found that RoBERTa (Liu et al., 2019) consistently performs better than BERT on *human-level* tasks (e.g., predicting

characteritics of the person that authored a text) including predicting the demographics (age, gender), personality (extraversion, openness) and mental health (suicide risk) of individual based on their text (Ganesan et al., 2021; Matero et al., 2021). For human level tasks, using the four last layers or the second to last layer have generally been found to yield good results; but recently Matero et al. (2022) more carefully examined which layer(s) that produce most accurate results in predicting depression related measures, which suggested that “layer 19 (sixth-to last) is the most ideal by itself” (p.1).

Limitations of transformers

Transformer models are very large. While researchers have developed ways to shrink models, the best versions of these models are still scaling exponentially. That is, the size of the files defining the models that users need are increasing more rapidly than the performance gains. While BERT-base comprises 110M parameters and achieves a Spearman rho of .86 on a standard semantic similarity task, BERT-large is approximately three times that size comprising 336M and achieves only a modest improvement in the same task: rho = .87. With costs of storage and memory fairly inexpensive today, it may often be more ideal to use the larger models at the expense of needing better computing equipment, but this could exclude users with more limited computing resources. Further, it is very expensive to create your own model.

Functions: mapping text to numbers (i.e., word embeddings)

Text includes three functions to map text to word embeddings. The textEmbed() is a high-level function that encompasses both *textEmbedLayersOutput()* and *textEmbedLayerAggregation()*. *textEmbedLayersOutput()* retrieves layers and hidden states from the language model; and *textEmbedLayerAggreation()* aggregates these layers to form word embeddings. It is possible to get both contextualized and decontextualized embeddings for individual words (i.e., where only one word at a time has been sent to the model).

textEmbed()

The textEmbed() function automatically transforms character variables in a given dataset (i.e., a dataframe or a tibble) to word embeddings. It is the main embedding function in *text*; and can output contextualized embeddings for text as well as decontextualized embeddings for each individual word (which may be used for the plotting functions, as described later).

The language model

The pretrained language model is set with the *model* setting in textEmbed() (or *textEmbedLayersOutput()*). The default model is "*bert-base-uncased"*, and other models can be specified by using their HuggingFace identifier (https://huggingface.co/models). When running a language model for the first time, the *text* package automatically downloads it using the python package *transformers*. The files necessary to run a model are cached in /Users/USER_NAME/.cache. Subsequent calls to text methods that use the same model will directly access the model in cache rather than downloading it again. The textModels() function lists models and tokenizers that you have downloaded, and textModelsRemove("model-name")

lists models and tokenizers that you have downloaded, and textModelsRemove('model-name') enables you to delete specific models and its accompanying files.

This tutorial uses example data that is accessible through the *text*-package. It is a subset from a study (see Kjell et al., 2019) where participants have described their satisfaction with life and harmony in life (or lack thereof) with a text response, 10 descriptive words or rating scales including the Satisfaction with Life Scale (SWLS; Diener et al., 1985) and the Harmony in Life Scale (HILS; Kjell et al., 2016). To embed text and build predictive models takes time. Embedding the included text examples takes approximately 10 minutes using a standard laptop. Word

embeddings and models for the tutorial have been saved as part of the open tutorial data, so it is possible to follow the tutorial without running the most time-consuming functions. (Note that it is possible to speed up the embedding process with access to graphics processing units; GPU).

```
# Look at example data included in the text package comprising both text and numerical
variables (note that there are only 40 participants in this example).
Language_based_assessment_data_8


# Transform the text/word data to word embeddings (see help(textEmbed) to see the default
settings).
word_embeddings <- textEmbed(Language_based_assessment_data_8,
model = "bert-base-uncased")

# See how the word embeddings are structured
word_embeddings

# Save the word embeddings to avoid having to embed the text again. It is good practice
to save output from analyses that take a lot of time to compute, which is often the case
when analyzing text data.
saveRDS(word_embeddings, "word_embeddings.rds")

# Get the saved word embeddings (again)
word_embeddings_saved <- readRDS("word_embeddings.rds")
```

Note that one of the output layers is referred to as layer 0, which is the original input embedding to BERT; and hence, it is not contextualized.

textEmbedLayersOutput()

The textEmbedLayersOutput() takes text as input, and returns the hidden states for each token of the text, including the [CLS] and the [SEP]. This gives you more control over the embedding process as compared to the textEmbed(). The output below shows the hidden states of layer 11 and 12 for each word/token of “I am fine”.

```
# Get hidden states for "I am fine"
imf_embeddings_11_12 <- textEmbedLayersOutput("I am fine",
                                              layers = 11:12,
                                              decontexts = FALSE)
imf_embeddings_11_12

# OUTPUT
```

```
$context
$context$x
$context$x[[1]]
# A tibble: 10 x 771
```

tokens	token_id	layer_number	Dim1	Dim2	Dim3
<chr>	<int>	<int>	<dbl>	<dbl>	<dbl>
[CLS]	1	11	0.266	-0.0910	-0.0521
i	2	11	0.178	-0.296	0.188
am	3	11	-0.114	-0.542	0.0757
fine	4	11	1.18	-0.434	-0.317
[SEP]	5	11	0.0461	0.0134	-0.0311
[CLS]	1	12	0.0609	0.377	0.213
i	2	12	-0.173	0.0227	0.0647
am	3	12	-0.406	-0.319	0.128
fine	4	12	0.763	-0.373	-0.00901
[SEP]	5	12	0.798	-0.0405	-0.324

```
# … with 765 more variables: Dim4 <dbl>, Dim5 <dbl>,Dim6 <dbl>, …
```

textEmbedLayerAggregation()

The textEmbedLayerAggregation() function aggregates the hidden states from textEmbedLayersOutput(). It is possible to select different methods for how layers and tokens are aggregated. The returned object is a word embedding that represents the entire text. Below we show how to 1) aggregate hidden states by concatenating layers 11 and 12, and computing the mean of each dimension across tokens (yielding a word embedding with 1,536 dimensions); and 2) only selecting layer 11 and computing the mean across tokens (yielding a word embedding with 768 dimensions).

```
# 1. Concatenate layers(results in 1,536 dimensions).
textEmbedLayerAggregation(imf_embeddings_11_12$context,
                          layers = 11:12,
                          aggregate_layers = "concatenate",
                          aggregate_tokens = "mean")
# OUTPUT
```

```
$x
# A tibble: 1 x 1,536

Dim1        Dim2        Dim3        Dim4        Dim5        Dim6
<dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>

0.311       -0.270      -0.0274     -0.408      0.202       0.113

… with 1,530 more variables: Dim7, Dim8, Dim9,...
```

```
# 2. Only select layer 11 (768 dimensions).
textEmbedLayerAggregation(imf_embeddings_11_12$context,
                          layers = 11,
                          aggregate_tokens = "mean")
# OUTPUT
```

```
$x
# A tibble: 1 x 768

Dim1        Dim2        Dim3        Dim4        Dim5        Dim6
<dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>

0.311       -0.270      -0.0274     -0.408      0.202       0.113

… with 762 more variables: Dim7, Dim8, Dim9,...
```

These word embeddings can be used for many different purposes and kinds of analyses. As such,

the *text*-package serves R-users as a point solution for transforming text to state-of-the-art word embeddings that are ready to be used for their specific tasks. The next part of the tutorial exemplifies how word embeddings may be used and how the *text*-package may serve as an end-to-end solution that provides AI techniques tailored for social and behavioral scientists.

Predictive Modeling with Word Embeddings

A common use of word embeddings is as input for statistical learning based predictive modeling (also known as machine learning). In such work for psychology, one often attempts to "predict" a score for a standard psychological assessment from language use (Park et al., 2015; Kjell et al., 2021). While there are endless types of predictive models from statistical learning in which one could input embeddings (see James et al., 2021 for a good introduction) numerous recent empirical studies have demonstrated that when using modern contextual embeddings, L2-penalized linear models yield state of the art results (Ganesan et al., 2021; Matero et al., 2019; Kjell et al., 2022). For example, Matero et al. (2019) used L2 regularized logistic regression on top of transformer-based contextual embeddings to achieve top results on the computational linguistics for psychology 2019 Shared Task B ("CLPsych-2019"; Zirikly et al., 2019), which compared 28 predictive systems, using techniques varying from convolutional neural networks (CNNs) to random forests or multi-layer perceptrons, from 11 participating teams. Shared tasks are particularly robust evaluations because the system designers are not giving the true labels but rather, they send their predictions based on text without labels to a 3rd party that runs the evaluation and reports the accuracies. Many other recent shared tasks have similarly confirmed systems with L2 penalized linear models on top of contextual embeddings to be among the best performing, such as a BERT-based system for the SemEval-2020 task on sentiment analysis (Palomino, & Ochoa-Luna, 2020)[2] or a system combining multiple contextual embeddings via an L2 penalized linear regression ("ridge regression") layerfor the SemEval-2020 task on humor detection (Hossain et al., 2020; Morishita et al., 2020) These works support the idea that because transformer-based contextual embeddings produce state-of-the-art semantic representations of text itself, sophisticated models on top of them are not necessary (and sometimes overfit) for producing accurate predictions (Bommasani, et al., 2021). Next, we describe this technique in more detail.

Word Embeddings as Input to a Predictive Model

The word embeddings may be used in predictive models such as multiple linear regression:

$$y = \beta_0 + \beta_1 * x_1 + \ldots + \beta_m * x_m + \varepsilon$$

where,

y	denotes the observed values (in which $\hat{y}$ represents the predicted value),
$x_1 - x_m$	the predictors, i.e., the dimensions of the word embeddings (where the subscript refers to the mth dimension),
β_0	the constant,

$\beta_1 - \beta_m$	the coefficients that define the relationship between embeddings and the outcome, and
ε	the error term.

Considering that the word embeddings often comprise many dimensions, it is often useful to first carry out a dimension reduction technique. To achieve this, Principal Component Analysis (e.g., see Wold et al., 1987) is implemented in *text* because we have noticed that it is particularly

[2] Note that in many modern NLP papers, an L2 penalization is referred to as “weight decay”

efficient for reducing the dimensionality of word embeddings. It is possible to select other statistical prediction models in *text*, including ridge regression (Hoerl & Kennard, 1970) or lasso regression (Tibshirani, 1996); and for categorical variables logistic regression or random forest (e.g., see Ho, 1995). To evaluate the predictions made by the statistical model, it is possible to correlate the predicted values with the observed values (i.e., cor[y, $\hat{y}$]) using cross-validation.

Cross Validation

Cross validation is a technique that may be used to evaluate and select a statistical model with minimal overfit (for a more detailed discussion on cross validation in psychology, e.g., see Mosier, 1951; Yarkoni & Westfall, 2017). Overfitting arises when data-specific errors or noise are modeled, which results in a model that is bad at generalizing to other data. The cross validation methods aim to limit this risk by estimating a model's predictive ability while simultaneously accounting for its ability to generalize (e.g., see Browne, 2000). The simplest form of a cross validation method randomly divides a dataset into a *training-set* and a *testing-set*. The parameters, i.e., the $\beta_1 \ -\beta_m$ in the specified regression above, are estimated in the training-set; and then applied to the test-set to predict values ($\hat{y}$) that can be compared with the observed values (y). Using this method results in an *out-of-sample performance*, where the trained model is applied on new data from the same population.

K-fold Cross Validation

K-fold cross validation is a procedure to divide the training and testing sets efficiently. If data is scarce, it may be considered wasteful to not use all data; e.g., to only use 50% of the data for training and 50% only for testing. First, it is possible to run the procedure again by alternating the purpose of the data-sets, so that the original testing-set becomes the training-set, and the original training-set becomes the testing-set: In this way all data will have predictions that can be correlated with the observed values. Second, in order to make better parameter estimations it is possible to get more power in the statistical models by making sure that the training set is larger. One common way is to select 90% of the data to the training set and 10% for the testing; and then alter the purpose of the data as previously described. This is done ten times, so that all cases get a prediction. Hence, k in k-fold refers to the number of groups (or folds); so the described procedure is called 10-fold cross validation. It is also possible to train on all but one case (which is called *leave-one-out* cross validation). Leave-one-out cross validation is good for very small dataset, whereas for larger dataset it requires a lot of computational power and time (for rather small gains).

Cross Validation with Test, Train and Development Sets

Cross-Validation with Test, Train and Development Sets

To fit models with hyperparameters (e.g., the penalty in ridge regression), a development set (also called assessment set) can be used to evaluate on. Figure 2 shows how the training set in the outside fold is split into a training (or analysis) set and development (assessment) set. Hence, the training set is used to fit models with different penalties; these are then assessed in the development set. The model with the best result (e.g., evaluated using the correlations between observed and predicted scores) is subsequently applied in the test set of the outside fold. This procedure is repeated for all folds.

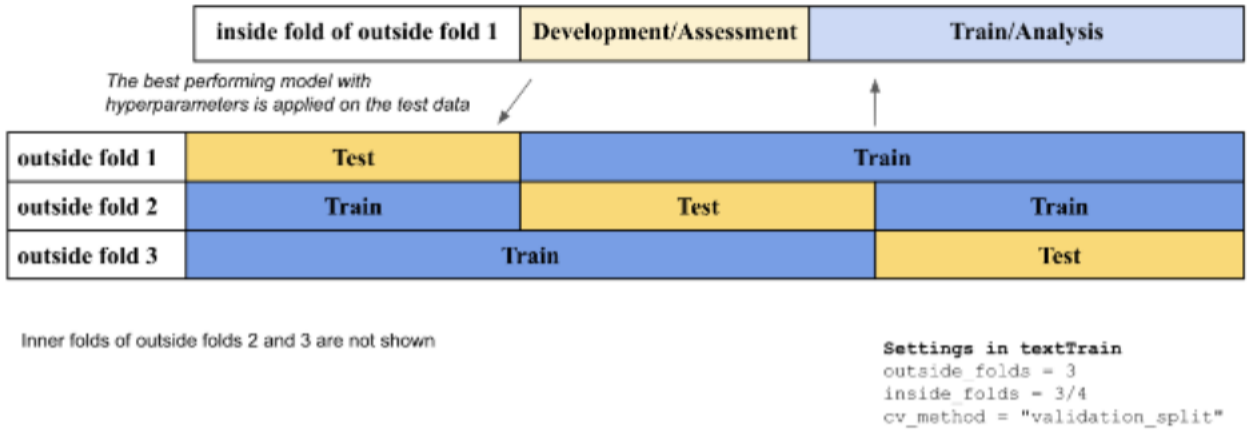
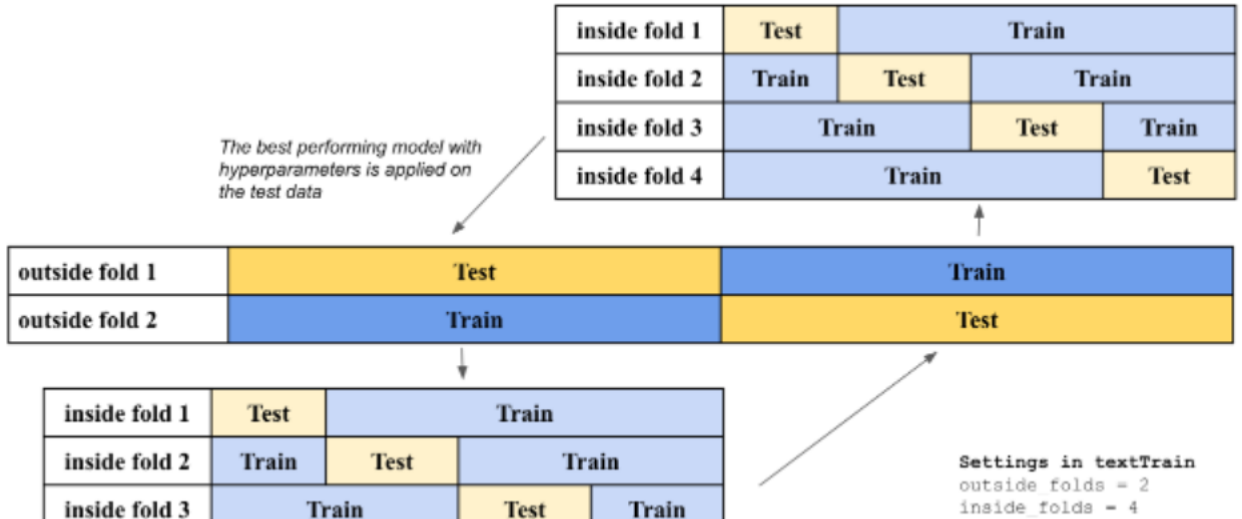


Figure 2. Cross Validation with 3 Outside Folds; and ¾ Analysis and ¼ Development Set

Nested K-Fold Cross Validation

To use nested cross validation, including train and test loops in the inner fold of each outside fold, is also a way to make better use of the data (see Figure 3). The above cross validation method with test, train and development sets focuses on achieving more accurate estimates of the machine learning approach; which is good when comparing machine learning algorithms. The nested k-fold cross validation focuses on achieving a more accurate model; but tends to take longer time because more models need to be fitted and evaluated.



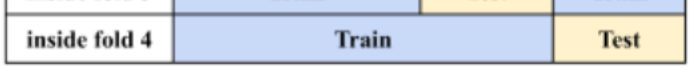


Figure 3. Example of Nested Cross Validation with 2 Outside Folds and 4 Inside Folds
Note that prioritizing more outside than inside folds makes often more sense.

Applying Predictive Models to New Data

Language-based predictive models can be applied to new data. Independently trained language models may be applied to new data to examine semantic-psychological features such as valence or arousal. It is also easy to share predictive models between projects and researchers. For example, it is possible to develop predictive models for valence and arousal, based on the Affective norms for English Words (ANEW; Bradley & Lang, 1999). The ANEW is a list of more than 1000 words that have been rated according to valence, dominance and arousal by participants. A valence model can be created by predicting valence from the word embeddings of the words in the list. First the model can be evaluated using cross validation to see how good it is; and then it can be saved without using cross validation to use all the data. This model can now be used to estimate the valence of any word or set of text represented by a word embedding based on the same language model (and settings) originally used to create the word embeddings in the predictive model. Hence, it is a flexible method considering that it is possible to get predictions for words that were not in the original model/words list, since the estimated parameters are applied to the dimensions of the word embeddings.

Functions

textTrain()

The textTrain() function is used to examine how well the word embeddings from a text can predict a numeric or categorical variable, based on the different cross-validation methods described above. The textTrain() function is a wrapper for textTrainRegression() and textTrainRandomForest(); by default, textTrain() uses the former for predicting numeric variables and the latter for predicting categorical variables. In the example below we examine how well the satisfaction text-responses can predict the rating scale scores from the Satisfaction with life scale. We advise researchers to share their predictive models, for example, on their open science framework account (www.osf.com) or GitHub. For this purpose it is possible to attach a description of the data and model by describing it in the describe_model setting. For example, use the following format:

> model_description = “author(s): XXX; data: N = XXX, population = XXX; publication: title = XXX; description: e.g., measure details etc.”

```
# Examine the relationship between satisfactiontext and the corresponding rating scale

model_satisfactiontext_swls <- textTrain(
  x = word_embeddings$satisfactiontexts, # the predictor variables (i.e., the word
embeddings)
  y = Language_based_assessment_data_8$swlstotal, # the criterion variable (i.e.,the
rating scale score.
  model_description = "author(s): Kjell, Giorgi, & Schwartz; data: N=40, population =
Online, Mechanical Turk; publication: title = Example for demo; description: swls =
the satisfaction with life scale")
```

```
# Examine the correlation between predicted and observed Harmony in life scale scores
model_satisfactiontext_swls$results

# OUTPUT:

Pearson's product-moment correlation
data:  predy_y$predictions and predy_y$y
t = 3.8766, df = 38, p-value = 0.0002032
alternative hypothesis: true correlation is greater than 0
95 percent confidence interval:
 0.3122273 1.0000000
sample estimates:
      cor
0.5323535

# Examine the names in the object returned from training
names(model_satisfactiontext_swls)

#OUTPUT:

[1] "predictions"
[2] "final_recipe"
[3] "final_model"
[4] "model_description"
[5] "results"
```

The returned output from training includes: 1) the cross-validated *predictions* with the observed values and row id; 2) the *final "recipe"* with information about how to preprocess the data before using it for training/prediction; 3) the *final model* with model information, which can be used for prediction, 4) the *model description* contain information about the options used to create the model as well as the information added by the user; and 5) the *results* including a comparison between cross-validated predictions and observed values/categories. The results for numeric values is by default a *Pearson* correlation, which can be changed with `method_cor`. The results for a classification task include a Fisher's Exact Test for Count Data, Pearson's Chi-squared test with Yates' continuity correction, a plotted ROC curve and metrics from the R-package *yardstick*

(Kuhn & Vaughan, 2020b) including accuracy, balanced accuracy (bal_accuracy), specificity (spec), sensitivity (sens), and kappa (kap).

textTrainLists()

The textTrainLists() runs through several text variables (i.e., lists of word embeddings) and/or numeric variables at the same time.

```
# Predicting several outcomes from several word embeddings
models_words_ratings <- textTrainLists(word_embeddings[1:2],
Language_based_assessment_data_8[5:6])

# See results
models_words_ratings$results


# OUTPUT
```

descriptions <chr>	correlation <chr>	df <chr>	p_value <chr>	t_statistics <chr>	alternative <chr>	p_value_corrected <dbl>
satisfactiontexts_hilstotal	0.46	38	0.0016	3.16	greater	3.13e-03
harmonytexts_hilstotal	0.68	38	8.56e-07	5.65	greater	3.43e-06
satisfactiontexts_swlstotal	0.53	38	0.0002	3.88	greater	6.10e-04
harmonytexts_swlstotal	0.39	38	0.0069	2.58	greater	6.89e-03

textPredict()

Trained models created by textTrain() can be applied to new datasets. In the next example we download and apply a model that has been trained to predict valence. This model was based on a list of nearly 14,000 words that were rated on valence, arousal and dominance (Warriner et al., 2013). To achieve the prediction we use the textPredict() function.

```
# Download and read a valence trained prediction model
# To download the model go to the following web address: https://r-
text.org/text_models/valence_Warriner_L11_12.rds
# Save the model to your working directory and run:
valence_Warriner_L11_12 <- readRDS("valence_Warriner_L11_12.rds")

# Examine the model
valence_Warriner_L11_12

# PART OF THE OUTPUT

        Pearson's product-moment correlation
```

```
data:  predy_y$predictions and predy_y$y
t = 130.17, df = 13913, p-value < 2.2e-16
alternative hypothesis: true correlation is greater than 0
95 percent confidence interval:
 0.7346788 1.0000000
sample estimates:
      cor
0.7410316
```

```
# Apply the model to the satisfaction text
satisfaction_text_valence <- textPredict(valence_Warriner_L11_12,
word_embeddings$satisfactiontexts)

# Examine the correlation between the predicted valence and the Satisfaction with life
scale score
psych::corr.test(satisfaction_text_valence$.pred,
Language_based_assessment_data_8$swlstotal)


# OUTPUT

Call:psych::corr.test(x = satisfaction_text_valence$.pred, y =
Language_based_assessment_data_8$swlstotal)
Correlation matrix
[1] 0.63
Sample Size
[1] 40
Probability values adjusted for multiple tests.
[1] 0
```

Computing Words or Texts for Semantic Similarity and Distance

Word embeddings may be used to measure how similar two words/texts are in meaning. The values composing a word embedding may be seen as coordinates in a high dimensional space; and the more similar two embedding vectors are the closer they are positioned in the space. There are many different ways to capture how closely positioned two vectors are; in *text* many common similarity and distance measures can be used, including “cosine”, "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski" (for a review of similarity measures see Chandrasekaran & Mago, 2021). The default measure in text, is to measure the cosine of the angle between them (see a simplified illustration in Figure 4). .

Like a correlation, the cosine can range from -1 to 1 (see Wickens, 2014, for a discussion on how correlation and cosine are mathematically related); but in the embedding space the cosine between two words is typically not much below 0. The smaller the angle, the higher the cosine is; and thus, the more similar are the embeddings (this cosine measure is referred to as a semantic similarity score).

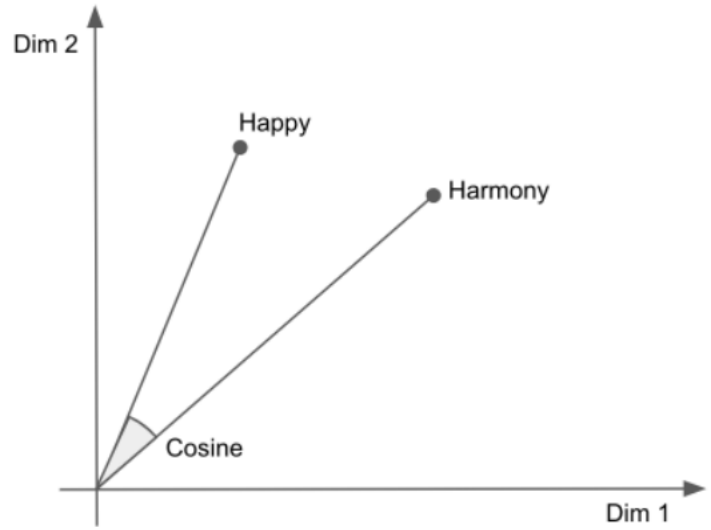


Figure 4. Illustration of the semantic similarity of two words in a simplified 2-dimensional embedding space (note that this also holds when adding many more dimensions).

The cosine may be seen as an unstandardized effect size indicating the strength of a relationship (Charikar, 2002); however, it should be noted that its absolute value is not comparable between different models and model specification setups. For example, Ethayarajh (2019) demonstrated that word embeddings have different distributions in different BERT layers, so that two random words on average tend to have higher cosine similarity in higher as opposed to lower layers. Thus, comparing two cosine based semantic similarity scores between two different models or layers is not interpretable; instead examine the relative difference in scores from the same model/layer(s).

Functions

textSimilarity()

The textSimilarity() function computes the semantic similarity between two text variables, and the textSimilarityNorm() computes the semantic similarity between one text variable and a word norm. Semantic similarity scores can, for example, be used to measure psychological constructs independent from rating scales by using word norms that represent the to-be-measured construct

(Kjell et al., 2019). A word norm can be created by asking participants to describe a psychological construct. The word norm can be used to measure the semantic similarity between a person's answer to whether they experience harmony in their life to the word norm describing harmony in life: If the score is high the person is seen to have high harmony in life. Below this analysis is described.

```
# Compute semantic similarity scores between two text columns, using the previously
created word_embeddings.
semantic_similarity_scores <- textSimilarity(word_embeddings$harmonytexts,
```

```
                                                word_embeddings$satisfactiontexts)
# Look at the first scores
head(semantic_similarity_scores)

# OUTPUT

# [1] 0.9227832 0.9143049 0.8953063 0.8512952 0.9411532 0.8952788


#  Read word norms text (later we will use these for the semantic centrality plot)
word_norms <- read.csv("Word_Norms_Mental_Health_Kjell2018_text.csv")


# Read the word embeddings for the word norms
word_norms_embeddings <-
readRDS("Word_Norms_Mental_Health_Kjell2018_text_embedding_L11_12.rds")

# Examine which word norms there are.
names(word_norms_embeddings)

# OUTPUT

[1] "harmonynorm"         "disharmonynorm"      "satisfactionnorm"
[4] "dissatisfactionnorm" "worrynorm"           "depressionnorm"
[7] "notatallworried"     "notatalldepressed"   "singlewords_we"



# Compute semantic similarity score between the harmony answers and the harmony norm
# Note that the descriptive word answers are used instead of text answers to
correspond with how the word norm was created.
norm_similarity_scores_harmony <- textSimilarityNorm(word_embeddings$harmonywords,
                                                     word_norms_embeddings$harmonynorm)

# Correlating the semantic measure with the corresponding rating scale
psych::corr.test(norm_similarity_scores_harmony,
Language_based_assessment_data_8$hilstotal)

# OUTPUT
Call:psych::corr.test(x = norm_similarity_scores_harmony, y =
Language_based_assessment_data_8$hilstotal)
Correlation matrix
[1] 0.6
Sample Size
[1] 40
```

```
Probability values adjusted for multiple tests.
[1] 0
```

Extended Functionality

In addition to implementations of standard analyses of text (the functions listed above), text comes with several less established novel functions. These are intended to support the core functionalities (and overtime become more empirically evaluated).

Testing the difference between two sets of texts

The *text similarity tests* compare whether the average meaning of two groups of words or texts significantly differ in meaning, using a permutation test approach. It can be compared with Semantic Textual Similarity (STS) tasks that seek to measure the degree of semantic equivalence between two snippets of text (e.g., see Agirre et al., 2016) and test of semantic difference (Arvidsson, Sikström, & Werbart, 2011). The text similarity tests compare the cosine similarity (or distance) between two groups, with a *permuted null distribution* that is created by randomly swapping the data between the two groups. The *p*-value is computed as the fraction of the values from the permuted null distribution that are at least as extreme as the observed cosine similarity statistic (i.e., from the non-permuted data). Importantly, the more permutations that are carried out, the more precise will the *p*-value be (for more information about permutation tests see Ludbrook & Dudley, 1998).

The proposed text similarity tests significance test the difference in meaning as represented within the word embedding space. As such, the tests focus on testing the difference in the latent "meaning" of the texts/words, as opposed to word count approaches that examine the relative frequency words occur in a text. Focusing on the meaning makes the tests powerful since similar words have a similar word embedding, whereas in word frequency tests different words but with similar meaning are counted as different. Lastly, the tests are carried out on a group level, making it possible to examine the difference between two sets of texts; however, it is not possible to significance test the difference between two single words. For example, it is not possible to test the difference between the words "happy" and "peaceful" on a single word level – like it is not possible to significance test the difference between two scores from a rating scale without knowing the underlying distribution of scores. The *text similarity tests* procedure for *paired* data is slightly different from the *unpaired* test procedure as described below.

Paired textSimilarityTest()

The *Paired textSimilarityTest()* examines whether there is a significant difference in meaning between two groups of texts in a within-subject design. It is achieved in three steps. First, the *Observed mean cosine statistic* is computed (see Table 1). This is achieved by calculating the mean of the absolute cosine values between paired responses (e.g., the cosine between HIL responses and SWL responses for all participants, and then take the mean of all participants):

> Observed mean cosine statistic = mean(|cos(Awe_{hil1}, Awe_{swl1})|, |cos(Awe_{hil2}, Awe_{swl2})|, …), where Awe is the aggregated word embedding from each of the responses; and "| |" means taking the absolute value.

Table 1.
Overview of the Text Difference Test based on Paired Permutations

	Response HIL		Response SWL		Cosine similarity between HIL and SWL	Mean
p1	we_{hil1} we_{hil2} ...	Awe_{hil}	p1	we_{swl1} we_{swl2} ...	z_{p1} = $\lvert\cos(Awe_{hil}, Awe_{swl})\rvert$	*Observed cosine statistic* (z_{all})= *Mean*(z_{p1}, z_{p2}, z_{p3})
p2	we_{hil1} we_{hil2} ...	Awe_{hil}	p2	we_{swl1} we_{swl2} ...	z_{p2} = $\lvert\cos(Awe_{hil}, Awe_{swl})\rvert$	
p3	we_{hil1} we_{hil2} ...	Awe_{hil}	p3	we_{swl1} we_{swl2} ...	z_{p3} = $\lvert\cos(Awe_{hil}, Awe_{swl})\rvert$	

Note. HIL = Harmony in life responses; SWL = Satisfaction with life response, p = Participant, we = word embedding, Awe = Aggregated word embedding; z = cosine similarity score.

Second, the permuted null distribution is created. Pairs of responses are created by randomly sampling from the HIL and SWL responses and then computing the cosine. As in step 1, the mean is computed. This procedure is repeated N (e.g., 10 000) times.

> Null distribution of permuted cosine statistic = mean(|cos($Awe_{random1}$, $Awe_{random1}$)|, |cos($Awe_{random2}$, $Awe_{random2}$)|, …) repeated N times, where random pairs are drawn from both sets.

Third, the observed cosine statistic (from step 1) is compared to the Null distribution of permuted cosine statistics (from step 2), to get the *p*-value. For a two tailed test this is achieved accordingly:

> 2 * min(sum(Permuted_values < Observed_value), sum(Permuted_values > Observed_value)) / length(Permuted_values).

Unpaired textSimilarityTest()

The *Unpaired textSimilarityTest()* examines whether there is a significant difference in meaning between two groups of texts in a between-subject design (e.g., between individuals taking part in an intervention versus those not taking part in an intervention). It is achieved in similar steps as described for the paired test, with the difference that the word embeddings are all aggregated for the entire group (not on an individual, participant level). First, the Observed cosine statistic is computed (see Table 2):

> $\cos(MAwe_{intervention.hil}, MAwe_{non\text{-}intervention.hil})$, where $MAwe_{intervention.hil}$ is the mean aggregated word embedding from all participants categorized as extravert, and $MAwe_{non\text{-}intervention.hil}$ is the mean aggregated word embedding from all participants categorized as introverts.

Table 2.
Overview of the Text Difference Test based on Unpaired Permutations

HIL response by intervention			HIL response by no-intervention			Cosine Similarity
p1	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{intervention.hil}$	Mean of Awe_{hil} $X_{intervention.hil}$	p4	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{no\text{-}intervention.hil}$	Mean of Awe_{hil} $X_{non\text{-}intervention,hil}$	*Observed cosine statistic* (z_{all}) = $cos(X_{intervention.hil}, X_{no\text{-}intervention.hil})$
p2	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{intervention.hil}$		p5	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{no\text{-}intervention.hil}$		
p3	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{intervention.hil}$		p6	we_{hil11} we_{hil12} ... we_{hil1n} $Awe_{no\text{-}intervention.hill}$		

Notes. HIL = Harmony in life responses; p = Participant, we = word embedding, Awe = Aggregated word embedding; z = cosine similarity score.

Second, the permuted null distribution is created:

> $\cos(X_{random1.hil}, X_{random1.hil})$, where $X_{random1.hil}$ is the aggregated word embedding from

$\cos(X_{random,int}, X_{random,int})$, where $X_{random,int}$ is the aggregated word embedding from embeddings randomly drawn from the intervention and non-intervention sample. The procedure is repeated N times.

Third, the statistics from Step 1 and 2 are compared using the formula provided in the paired test above.

Functions

textSimilarityTest()

The textSimilarityTest() function examines whether word embeddings of two groups of texts statistically differ in meaning. Below we compare the meaning between individuals' harmony in life and satisfaction with life answers using a paired test.

```
# Test the semantic difference between the satisfaction with life texts and the
harmony in life text.
s_test <- textSimilarityTest(word_embeddings$satisfactiontexts,
                             word_embeddings$harmonytexts,
                             method = "paired",
                             Npermutations = 100)

s_test

# PART OF OUTPUT
$p.value
[1] 0.00990099
```

Words' Position in the Embedding Space

There are many different ways to visualize the words in a data set such as multidimensional scaling (MDS; e.g., Borg & Groenen, 2005), t-Distributed Stochastic Neighbor Embedding (t-SNE; Maaten, & Hinton, 2008) and Principal Component Analysis (PCA; see Appendix I for the text functions textPCA and textPCAPlot). Here we will briefly demonstrate two plot functions in *text for demonstrating potential basic uses of the word embeddings*: The Supervised Dimension Projection Plot shows words that are significantly related to one as compared to another group, where the test statistics are tested in permutation test procedures. The Semantic Centrality Plot shows the words that are most semantically central to a set of texts/words. Both functions plot the words' position based on their word embeddings.

The Supervised Dimension Projection

The Supervised Dimension Projection

The Supervised Dimension Projection compares two groups (e.g., intervention versus non-intervention participant answers), responses to different questions (e.g., harmony versus satisfaction responses), or low versus high scorers on a rating scale using median split or lower/higher quartiles. In short, we construct an embedding that captures the difference between the two groups, a vector that forms a line through the origin (the *aggregated direction embedding*); then all individual words are “projected” onto that direction line (where dot product is computed to

"project" a vector onto another vector; see Figure 5 for a visualization). More precisely, the plot is based on the following steps:

Preprocessing

1. Responses are divided into two groups (G1 and G2; where a scale variable is split according to mean or lower and higher quartile).

2. The aggregated word embeddings of the two groups are computed: *The G1 split aggregation embedding* and *the G2 split aggregation embedding*.

Supervised Dimension Projection Statistics

3. An *Aggregated direction embeddings* is computed:
   Aggregated direction embeddings =
   G2 split aggregation embedding - G1 split aggregation embedding

   So, for example, the direction of harmony = Group(high harmony) - Group(low harmony); where the direction is seen to go through the origin and the aggregated direction embedding.

4. All individual word embeddings are positioned (or anchored) to the same point. So, for each word:

   Anchored embedding = original embedding - aggregation embedding of G1 and G2 from step 2.

5. To project onto the *Aggregated direction embeddings* (i.e., from step 3.), the dot product is computed between the *Anchored embedding* of all individual words (i.e., from step 4) and the *Aggregated direction embeddings* (i.e., step 3). That is, dot product(Anchored we , Aggregated direction we) = a point on the direction.

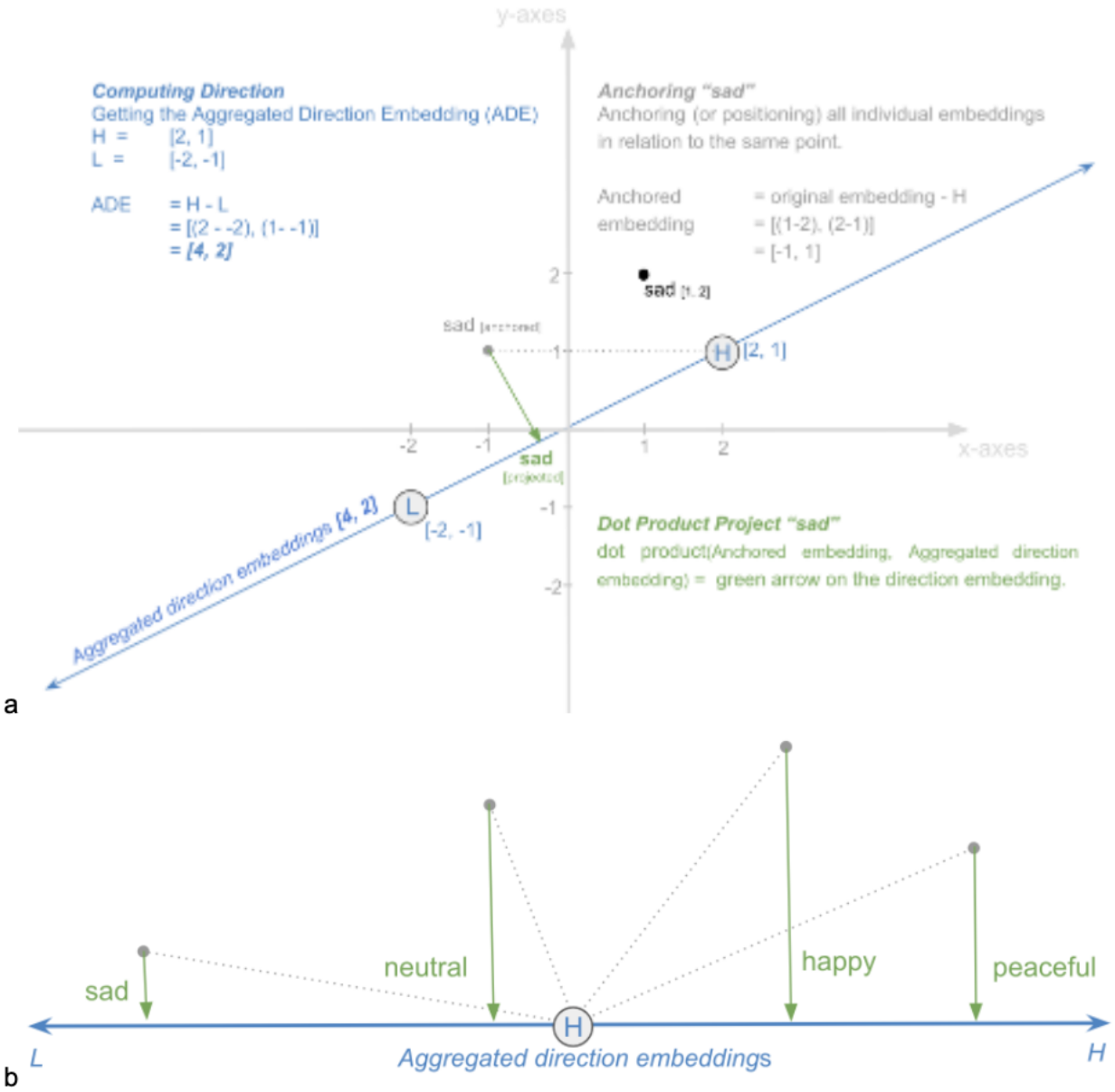


Figure 5ab. Illustrations of the Supervised Dimension Projection Method The illustration exemplifies the reduction of two dimensions to one dimension, where in practice there are many more dimensions.

Computing *p*-values with a permutation procedure

6. A permuted null distribution of Supervised Dimension Projections is created by computing the dot product between randomly selected word embeddings from G1 and G2, and a Permuted aggregated direction embeddings (i.e., the direction embedding is also created by randomly swapping words from G1 and G2).

7. *P*-values for each word are computed by comparing their Supervised Dimension Projection with the permuted null distribution of Supervised Dimension Projections from step 6, as in previous descriptions; whilst correcting for multiple comparisons according to selected methods.

Functions

The plotting is made with two functions, in two steps: First, *textProjection()* analyzes the data, including Supervised Dimension Projections, null distributions, *p*-values, and frequencies. Second, textProjectionPlot() uses the output from *textProjection()* to plot the words, including providing design options for the figure. Dividing the plotting procedure into these two steps makes the process more transparent (i.e., the user naturally sees the output according to which the words will be plotted) as well as being more flexible by enabling the user to try out different visual settings without the need to rerun the heavy computational processes.

textProjection()

```
library(tidyverse)
# Merge satisfaction and harmony words to one column
worddata_merged <- c(Language_based_assessment_data_8$satisfactionwords,
                     Language_based_assessment_data_8$harmonywords)

# Create variable indicating whether it is a satisfaction or harmony word response
satwor1_harwor2                                <-                                c(rep(1,
length(Language_based_assessment_data_8$satisfactionwords)),
                     rep(2, length(Language_based_assessment_data_8$harmonywords)))

# Replicating the SWLS score so that it has equal length as the merged words and
satwor1_harwor2
swls_scores <- c(Language_based_assessment_data_8$swlstotal,
                 Language_based_assessment_data_8$swlstotal)

# Merge satisfaction and harmony embeddings
embedding_merged <- bind_cols(word_embeddings$satisfactionwords,
                              word_embeddings$harmonywords)


# Pre-processing data for plotting
projection_results <- textProjection(worddata_merged,
                                     embedding_merged,
```

```
                                word_embeddings$singlewords_we,
                                satwor1_harwor2,
                                swls_scores
)
projection_results$word_data
 # PART OF OUTPUT
```

words <chr>	dot.x <dbl>	p_values_dot.x <dbl>	n_g1.x <dbl>	n_g2.x <dbl>
accepted	-0.681	.144	-1	*NA*
accepting	0.140	.886	*NA*	2
accomplished	-2.04	.0001	-2	*NA*
achievement	-0.176	.614	-1	*NA*
active	-0.905	.0875	-1	*NA*
adequate	-0.161	.642	-1	*NA*

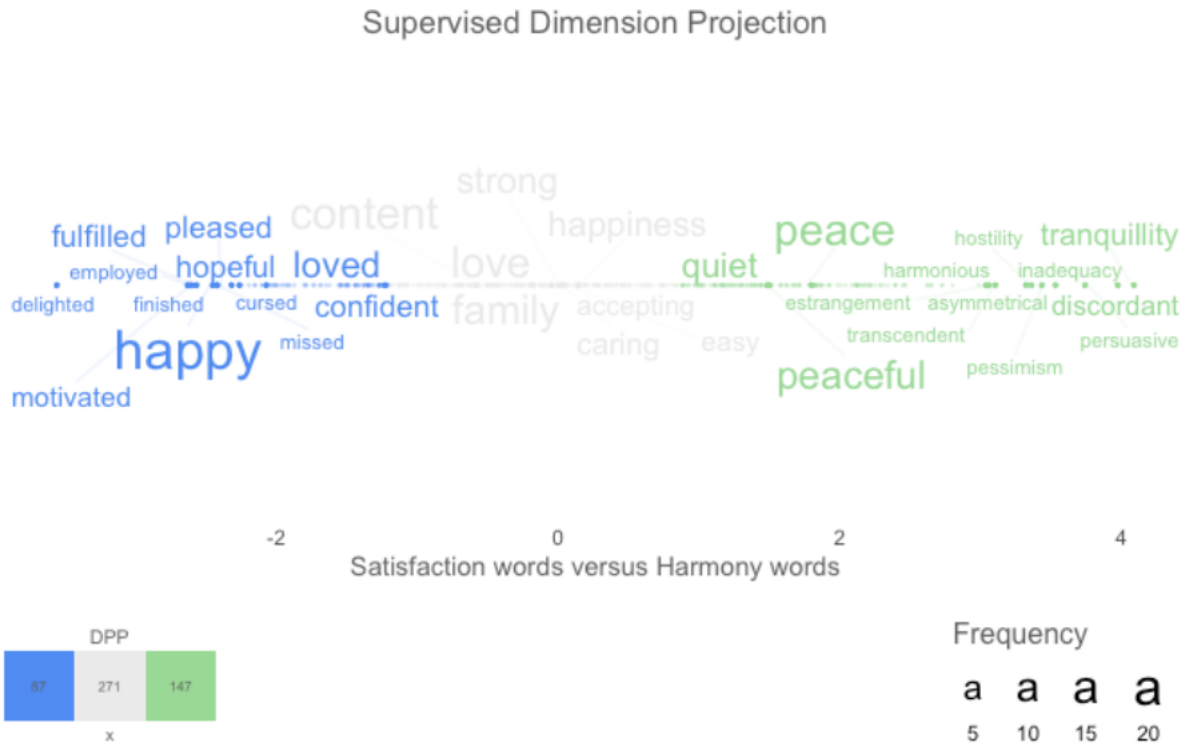
textProjectionPlot()

```
# Supervised Dimension Projection Plot
# To avoid warnings -- and that words do not get plotted, first increase the max.overlaps
for the entire session:
options(ggrepel.max.overlaps = 1000)

plot_projection <- textProjectionPlot(
  word_data = projection_results,
  min_freq_words_plot = 1,
  plot_n_word_extreme = 10,
  plot_n_word_frequency = 5,
  plot_n_words_middle = 5,
  y_axes = FALSE,
  p_alpha = 0.05,
  p_adjust_method = "fdr",
  title_top = "Supervised Dimension Projection",
  x_axes_label = "Satisfaction words versus Harmony words",
  y_axes_label = "",
  bivariate_color_codes = c("#60A1F7", "#5DC688", "#40DD52",
                            "#398CF9", "#EAEAEA", "#85DB8E",
                            "#E07f6a", "#FF0000", "#EA7467")
)
# View plot
plot_projection$final_plot

#OUTPUT
```

```
```



DPP = Dot Product Projection. The transparent points in the figure show the distribution of words that are *not* plotted, whereas the non-transparent points indicate the position of the plotted words.

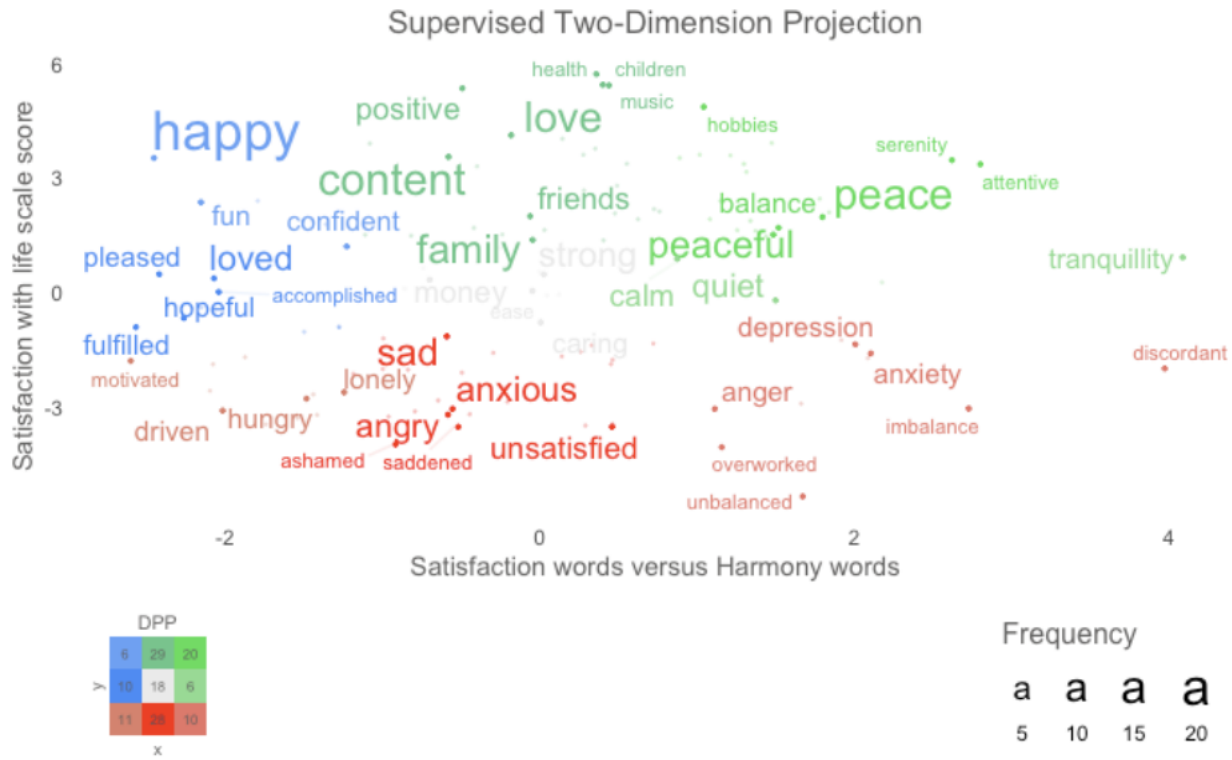
```

```
# Supervised Dimension Projection Plot
plot_projection_2D <- textProjectionPlot(
  word_data = projection_results,

  y_axes = TRUE,

  p_adjust_method = "fdr",
  title_top = "Supervised Two-Dimension Projection",
  x_axes_label = "Satisfaction words versus Harmony words",
  y_axes_label = "Satisfaction with life scale score",
```
```

```
  bivariate_color_codes = c("#60A1F7", "#5DC688", "#40DD52",
                            "#398CF9", "#EAEAEA", "#85DB8E",
                            "#E07f6a", "#FF0000", "#EA7467")
)
# View plot
plot_projection_2D$final_plot
```



Footnote. A supervised dimension projection plot of words significantly differing between responses to the satisfaction with life and the harmony in life (x-axis) and the level of satisfaction with life scale score (y-axis). The font size of the words indicates their frequency. The color indicates whether a word is significant or not significant (gray) when correcting for multiple comparisons using the false discovery rate (FDR). The color-legend in the lower left corner indicates the color and number of significant words in each part of the figure (for example, there are 20 light green words that are significantly high on both the x-axis and y-axis). The non-transparent dots represent the point for visible words, whereas transparent dots show the position of words that are not presented in the figure. The values on the x- and y-axes represent the dot product projection value (these should be compared with caution between figures); DPP = Dot product projection.

Words' Semantic Centrality

The Semantic Centrality Plot aims to highlight words that are semantically similar to the aggregated word embeddings of all words in the given text variable. Hence, it describes the

psychological construct or latent meaning of a text under investigation in the word embedding space. The aim is to highlight words from one type of response, rather than comparing two groups/dimensions. The statistics are computed with the textCentrality() functions. This is achieved in the following steps:

1. Computing the *Aggregated word embedding* (Awe) based on all words.

2. Computing the observed semantic similarity scores between individual word's embeddings (Iwe) and the aggregated word embedding (Awe).

   cos(Iwe, Awe) = semantic similarity score

Using the textCentralityPlot() it is possible to select the most extreme semantic centrality scores, middle scores as well as selecting words based on their frequency.

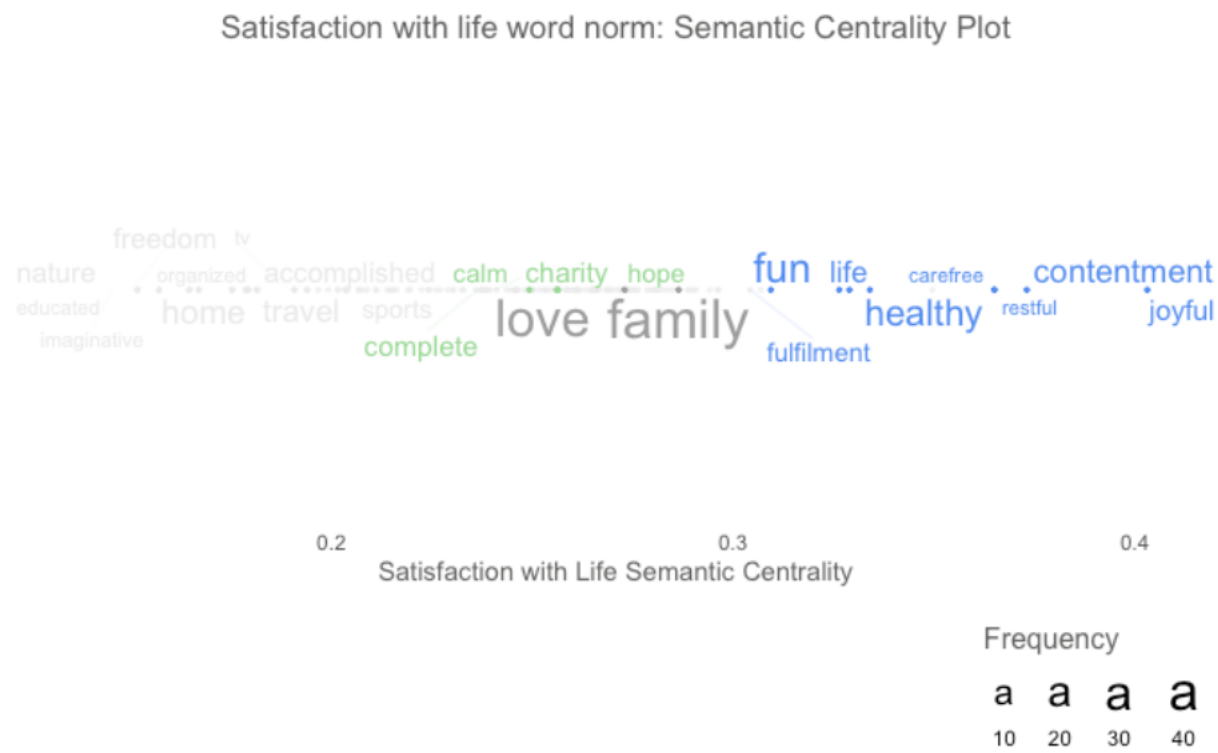
Functions

```
# Computing words' centrality (semantic similarity) score to the aggregated embedding
of all words
centrality_results <- textCentrality(words = word_norms$satisfactionnorm,
                                     word_embeddings                                    =
word_norms_embeddings$satisfactionnorm,
                                     word_norms_embeddings$singlewords_we)

centrality_plot <- textCentralityPlot(word_data=centrality_results,
                                      min_freq_words_test = 2,
                                      plot_n_word_extreme = 10,
                                      plot_n_word_frequency = 5,
                                      plot_n_words_middle = 5,
                                      title_top = "Satisfaction with life word norm:
Semantic Centrality Plot",
                                      x_axes_label = "Satisfaction with Life Semantic
Centrality")

centrality_plot$final_plot

# OUTPUT
```



Footnote. A Semantic Centrality Plot illustrating the words composing the Satisfaction with life word norm. The size of the words represents their frequency; the color their position. The values on the x-axes represent the semantic similarity scores (measured as the cosine score) between each word's embedding and the aggregated word embedding of all words in the word norm. The non-transparent dots represent the point for visible words, whereas transparent dots show the position of words that are not presented in the figure.

Summary

The *text* package has two main aims: First, to work as a modular solution for transforming text to state-of-the-art word embeddings for R-users. Second, to work as an end-to-end solution focusing on relevant functions for social sciences and human-level analyses. It is our hope that the *text* package can increase the ability of psychological scientists to analyze, with state-of-the-art computational techniques, one of the fundamental and key behaviors of people: natural language.

R-packages

R-packages used in text include:
dplyr (Wickham et al., 2020), tokenizers (Mullen et al., 2018), psych (Revelle, 2019), tibble (& Wickham, 2020), stringr (Wickham, 2019), tidyr (Wickham & Henry, 2020), ggplot2 (Wickham, 2016), ggrepel (Slowikowski, 2020), cowplot (Wilke, 2019), scales (Wickham & Seidel, 2020), rlang (Henry & Wickham, 2020b), purrr (Henry & Wickham, 2020a), Matrix (Bates & Maechler, 2019), stringi (Gagolewski, 2020), data.table (Dowle & Srinivasan, 2019), magrittr (Bache &

Wickham, 2014), parsnip (Kuhn & Vaughan, 2020a), recipes (Kuhn & Wickham, 2020), reticulate (Ushey et al., 2020), rsample (Kuhn et al., 2020), tune (Kuhn, 2020), workflows (Vaughan, 2020), yardstick (Kuhn & Vaughan, 2020b), quanteda (Benoit et al., 2018), broom (Robinson & Hayes, 2020), knitr (Xie, 2014), rmarkdown (Xie et al., 2018), testthat (Wickham, 2011), and rio (Chan et al., 2018).

Python libraries

Python packages integrated within the *text*-package include:
PyTorch (Paszke et al., 2019), transformers (Wolf et al., 2019), nltk (Bird, Klein, & Loper, 2009) and numpy (Oliphant, 2006).

References

Arvidsson, D., Sikström, S., & Werbart, A. (2011). Changes in self and object representations following psychotherapy measured by a theory-free, computational, semantic space method. Psychotherapy Research, 21(4), 430-446.

Agirre, E., Banea, C., Cer, D., Diab, M., Gonzalez Agirre, A., Mihalcea, R., ... & Wiebe, J. (2016). Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In SemEval-2016. 10th International Workshop on Semantic Evaluation; 2016 Jun 16-17; San Diego, CA. Stroudsburg (PA): ACL; 2016. p. 497-511.. ACL (Association for Computational Linguistics).

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Jin, D., Naumann, T., & McDermott, M. (2019). Publicly available clinical BERT embeddings. *ArXiv Preprint ArXiv:1904.03323*.

Andersen, N., & Zehner, F. (2021). shinyReCoR: A Shiny Application for Automatically Coding Text Responses Using R. *Psych*, *3*(3), 422–446. https://doi.org/10.3390/psych3030030

Babić, K., Martinčić-Ipšić, S., & Meštrović, A. (2020). Survey of Neural Text Representation Models. *Information*, *11*(11), 511. https://doi.org/10.3390/info11110511

Bache, S. M., & Wickham, H. (2014). *magrittr: A Forward-Pipe Operator for R*. https://CRAN.R-project.org/package=magrittr

Bates, D., & Maechler, M. (2019). *Matrix: Sparse and Dense Matrix Classes and Methods*. https://CRAN.R-project.org/package=Matrix

Beltagy, I., Lo, K., & Cohan, A. (2019). SciBERT: A pretrained language model for scientific text. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language*

Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 3606–3611.

Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S., & Matsuo, A. (2018). quanteda: An R package for the quantitative analysis of textual data. *Journal of Open*

Source Software, *3*(30), 774. https://doi.org/10.21105/joss.00774

Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. Reilly Media, Inc.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*(Jan), 993–1022.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., & Brunskill, E. (2021). On the opportunities and risks of foundation models. *ArXiv Preprint ArXiv:2108.07258*.

Borg, I., & Groenen, P. J. (2005). Modern multidimensional scaling: Theory and applications. Springer Science & Business Media.

Bradley, M. M., & Lang, P. J. (1999). *Affective norms for English words (ANEW): Instruction manual and affective ratings*. Technical Report C-1, The Center for Research in Psychophysiology, University of Florida.

Bratt J., & Harmon J. (2020). RBERT: R Implementation of BERT. R package version 0.1.11. https://github.com/jonathanbratt/RBERT.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A. (2020). Language models are few-shot learners. *ArXiv Preprint ArXiv:2005.14165*.

Browne, M. W. (2000). Cross-validation methods. *Journal of Mathematical Psychology*, *44*(1), 108–132.

Campbell, R. S., & Pennebaker, J. W. (2003). The secret life of pronouns flexibility in writing style

and physical health. *Psychological Science*, *14*(1), 60–65.

Carlson, T. A., Simmons, R. A., Kriegeskorte, N., & Slevc, L. R. (2014). The emergence of semantic meaning in the ventral temporal pathway. *Journal of Cognitive Neuroscience*, *26*(1), 120–131.

Chan, C., Chan, G. C., Leeper, T. J., & Becker, J. (2018). *rio: A Swiss-army knife for data file I/O*.

Chandrasekaran, D., & Mago, V. (2021). Evolution of semantic similarity—a survey. ACM Computing Surveys (CSUR), 54(2), 1-37.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, 380–388.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv Preprint ArXiv:1810.04805*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Diener, E., Emmons, R. A., Larsen, R. J., & Griffin, S. (1985). The satisfaction with life scale. *Journal of Personality Assessment*, *49*(1), 71–75.

Dougal, S., & Rotello, C. M. (2007). “Remembering” emotional words is based on response bias, not recollection. *Psychonomic Bulletin & Review*, *14*(3), 423–429. https://doi.org/10.3758/bf03194083

Dowle, M., & Srinivasan, A. (2019). *data.table: Extension of `data.frame`*. https://CRAN.R-project.org/package=data.table

Eichstaedt, J. C., Kern, M. L., Yaden, D. B., Schwartz, H. A., Giorgi, S., Park, G., Hagan, C. A., Tobolsky, V., Smith, L. K., & Buffone, A. (2020). Closed-and Open-Vocabulary Approaches to Text Analysis: A Review, Quantitative Comparison, and Recommendations. *Psychological Methods*.

Eichstaedt, J. C., Schwartz, H. A., Kern, M. L., Park, G., Labarthe, D. R., Merchant, R. M., Jha, S., Agrawal, M., Dziurzynski, L. A., & Sap, M. (2015). Psychological language on Twitter predicts county-level heart disease mortality. *Psychological Science*, *26*(2), 159–169.

Eichstaedt, J. C., Smith, R. J., Merchant, R. M., Ungar, L. H., Crutchley, P., Preoţiuc-Pietro, D., Asch, D. A., & Schwartz, H. A. (2018). Facebook language predicts depression in medical records. *Proceedings of the National Academy of Sciences*, *115*(44), 11203–11208.

Erk, K. (2012). Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, *6*(10), 635–653.

Ethayarajh, K. (2019). How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings. *ArXiv Preprint ArXiv:1909.00512*.

Feinerer, I., & Hornik, K. (2019). *tm: Text Mining Package*. https://CRAN.R-project.org/package=™

Firth, J. R. (1957). A synopsis of linguistic theory 1930– 1955. In Studies in Linguistic Analysis. Philological Society. Reprinted in Palmer, F. (ed.) 1968. Selected Papers of J. R. Firth. Longman, Harlow.

Flake, J. K., Pek, J., & Hehman, E. (2017). Construct Validation in Social and Personality Research: Current Practice and Recommendations. *Social Psychological and Personality Science*. https://doi.org/10.1177/1948550617693063

Gagné, C. L., Spalding, T. L., & Ji, H. (9). Re-examining evidence for the use of independent relational representations during conceptual combination. *Journal of Memory and Language*, *53*(3), 445–455. https://doi.org/10.1016/j.jml.2005.03.006

Gagolewski, M. (2020). *R package stringi: Character string processing facilities*. http://www.gagolewski.com/software/stringi/

Ganesan, A. V., Matero, M., Ravula, A. R., Vu, H., & Schwartz, H. A. (2021). Empirical Evaluation of Pre-trained Transformers for Human-Level NLP: The Role of Sample Size and Dimensionality. *ArXiv Preprint ArXiv:2105.03484*.

Garcia, D., & Sikström, S. (2013). Quantifying the Semantic Representations of Adolescents' Memories of Positive and Negative Life Events. *Journal of Happiness Studies*, *14*(4), 1309–1323. a9h. https://doi.org/10.1007/s10902-012-9385-8

Golub, G., & Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, *2*(2), 205–224.

Grolemund, G., & Wickham, H. (2018). *R for data science*.

Henry, L., & Wickham, H. (2020a). *purrr: Functional Programming Tools*. https://CRAN.R-project.org/package=purrr

Henry, L., & Wickham, H. (2020b). *rlang: Functions for Base Types and Core R and "Tidyverse" Features*. https://CRAN.R-project.org/package=rlang

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, *1*, 278–282.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hoerl, A. E., & Kennard, R. W. %J T. (1970). *Ridge regression: Biased estimation for nonorthogonal problems*. *12*(1), 55–67.

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

Hossain, N., Krumm, J., Gamon, M., & Kautz, H. (2020). Semeval-2020 Task 7: Assessing humor in edited news headlines. arXiv preprint arXiv:2008.00304.

Iliev, R., Dehghani, M., & Sagi, E. (2014). Automated text analysis in psychology: Methods,

applications, and future developments. *Language and Cognition*, *7*(2), 265–290. https://doi.org/10.1017/langcog.2014.30.

James, G., Witten, D., Hastie, T., Tibshirani, R., Sohil, F., Sohali, M. U., & Shabbir, J. (2021). An introduction to statistical learning with applications in R.

Jawahar, G., Sagot, B., & Seddah, D. (2019). What does BERT learn about the structure of language? *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

Jurafsky D., & Martin J. (2020). Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf.

Kjell, K., Johnsson, P., & Sikström, S. (2021). Freely Generated Word Responses Analyzed With Artificial Intelligence Predict Self-Reported Symptoms of Depression, Anxiety, and Worry. Frontiers in Psychology, 12.

Kjell, O. N. E., Daukantaitė, D., Hefferon, K., & Sikström, S. (2016). The Harmony in Life Scale Complements the Satisfaction with Life Scale: Expanding the Conceptualization of the Cognitive Component of Subjective Well-Being. *Social Indicators Research*, *126*(2), 893–919. https://doi.org/10.1007/s11205-015-0903-z

Kjell, O.N.E., Giorgi S., and Schwartz H.A. (2022) Dataset, code and models for the text package tutorial. DOI /10.17605/OSF.IO/DGCZT

Kjell, O. N., Kjell, K., Garcia, D., & Sikström, S. (2019). Semantic measures: Using natural language processing to measure, differentiate, and describe psychological constructs. *Psychological Methods*, *24*(1), 92.

Kjell, O. N., Sikström, S., Kjell, K., & Schwartz, H. A. (2022). Natural language analyzed with AI-based transformers predict traditional subjective well-being measures approaching the theoretical upper limits in accuracy. *Scientific reports*, 12(1), 1-9.

Kuhn, M. (2020). *tune: Tidy Tuning Tools*. https://CRAN.R-project.org/package=tune

Kuhn, M., Chow, F., & Wickham, H. (2020). *rsample: General Resampling Infrastructure*. https://CRAN.R-project.org/package=rsample

Kuhn, M., & Vaughan, D. (2020a). *parsnip: A Common API to Modeling and Analysis Functions*.

https://CRAN.R-project.org/package=parsnip

Kuhn, M., & Vaughan, D. (2020b). *yardstick: Tidy Characterizations of Model Performance*. https://CRAN.R-project.org/package=yardstick

Kuhn, M., & Wickham, H. (2020). *recipes: Preprocessing Tools to Create Design Matrices*. https://CRAN.R-project.org/package=recipes

Lauriola, I., Lavelli, A., & Aiolli, F. (2022). An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools. *Neurocomputing*, *470*, 443–456. https://doi.org/10.1016/j.neucom.2021.05.103

Libovický, J., Rosa, R., & Fraser, A. (2019). How Language-Neutral is Multilingual BERT? *ArXiv Preprint ArXiv:1911.03310*.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, *22 140*, 55.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *ArXiv Preprint ArXiv:1907.11692*.

Lynn, V., Balasubramanian, N., & Schwartz, H. A. (2020, July). Hierarchical modeling for user personality prediction: The role of message-level attention. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (pp. 5306-5316).

Matero, M., Hung, A., & Schwartz, H. A. (2021). Understanding RoBERTa's Mood: The Role of Contextual-Embeddings as User-Representations for Depression Prediction. *ArXiv Preprint ArXiv:2112.13795*.

Matero, M., Idnani, A., Son, Y., Giorgi, S., Vu, H., Zamani, M., ... & Schwartz, H. A. (2019, June). Suicide risk assessment with multi-level dual-context language and BERT. In Proceedings of the sixth workshop on computational linguistics and clinical psychology (pp. 39-44).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations

of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 3111–3119.

Morishita, T., Morio, G., Ozaki, H., & Miyoshi, T. (2020, December). Hitachi at SemEval-2020 task 7: Stacking at scale with heterogeneous language models for humor recognition. In Proceedings of the Fourteenth Workshop on Semantic Evaluation (pp. 791-803).

Mosier, C. I. (1951). I. Problems and designs of cross-validation 1. *Educational and Psychological Measurement*, *11*(1), 5–11.

Mullen, L. A., Benoit, K., Keyes, O., Selivanov, D., & Arnold, J. (2018). Fast, Consistent Tokenization of Natural Language Text. *Journal of Open Source Software*, *3*(23), 655. https://doi.org/10.21105/joss.00655

Müller, K., & Wickham, H. (2020). *tibble: Simple Data Frames*. https://CRAN.R-project.org/package=tibble

Nayak P. (2019). Understanding searches better than ever before. Retrieved at: https://blog.google/products/search/search-language-understanding-bert/.

Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA.

Osgood, C. E., Suci, G. J., & Tannenbaum, P. H. (1957). *The measurement of meaning*. University of Illinois press.

Otter, D. W., Medina, J. R., & Kalita, J. K. (2019). A Survey of the Usages of Deep Learning in Natural Language Processing. *ArXiv:1807.10854 [Cs]*. http://arxiv.org/abs/1807.10854

Palomino, D., & Ochoa-Luna, J. (2020). Palomino-Ochoa at SemEval-2020 Task 9: Robust system based on transformer for code-mixed sentiment classification. arXiv preprint

arXiv:2011.09448.

Park, G., Schwartz, H. A., Eichstaedt, J. C., Kern, M. L., Kosinski, M., Stillwell, D. J., Ungar, L. H., & Seligman, M. E. P. (2014). Automatic Personality Assessment Through Social Media Language. *Journal of Personality and Social Psychology*. pdh. https://doi.org/10.1037/pspp0000020

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained Models for Natural Language Processing: A Survey. *Science China Technological Sciences*, *63*(10), 1872–1897. https://doi.org/10.1007/s11431-020-1647-3

R Core Team. (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Revelle, W. (2019). *psych: Procedures for Psychological, Psychometric, and Personality Research*. Northwestern University. https://CRAN.R-project.org/package=psych

Robinson, D., & Hayes, A. (2020). *broom: Convert Statistical Analysis Objects into Tidy Tibbles*. https://CRAN.R-project.org/package=broom

Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). A primer in bertology: What we know about how bert works. *ArXiv Preprint ArXiv:2002.12327*.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *ArXiv Preprint ArXiv:1910.01108*.

Sikström, S., Kjell, O. N. E., & Kjell, K. (2018). *Semantic Excel: An Introduction to a User-Friendly Online Software Application for Statistical Analyses of Text Data* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/z9chp

Silge, J., & Robinson, D. (2016). tidytext: Text Mining and Analysis Using Tidy Data Principles in R. *JOSS*, *1*(3). https://doi.org/10.21105/joss.00037

Slowikowski, K. (2020). *ggrepel: Automatically Position Non-Overlapping Text Labels with "ggplot2."* https://CRAN.R-project.org/package=ggrepel

Son, Y., Clouston, S. A., Kotov, R., Eichstaedt, J. C., Bromet, E. J., Luft, B. J., & Schwartz, H. A. (2020). World Trade Center responders in their own words: Predicting PTSD symptom trajectories with AI-based language analyses of interviews. *ArXiv Preprint ArXiv:2011.06457*.

Sun, J., Schwartz, H. A., Son, Y., Kern, M. L., & Vazire, S. (2020). The language of well-being: Tracking fluctuations in emotion experience through everyday speech. *Journal of Personality and Social Psychology*, *118*(2), 364.

Tausczik, Y. R., & Pennebaker, J. W. (2010). The psychological meaning of words: LIWC and computerized text analysis methods. *Journal of Language and Social Psychology*, *29*(1), 24–54.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.

Ushey, K., Allaire, J. J., & Tang, Y. (2020). *reticulate: Interface to "Python."* https://CRAN.R-project.org/package=reticulate

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).

Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, \Lukasz, &

Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.

Vaughan, D. (2020). *workflows: Modeling Workflows*. https://CRAN.R-project.org/package=workflows

Warriner, A. B., Kuperman, V., & Brysbaert, M. (2013). Norms of valence, arousal, and dominance

for 13,915 English lemmas. *Behavior Research Methods*, *45*(4), 1191–1207. https://doi.org/10.3758/s13428-012-0314-x

Wickens, T. D. (2014). *The geometry of multivariate statistics*. Psychology Press.

Wickham, H. (2011). testthat: Get Started with Testing. *The R Journal*, *3*, 5–10.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H. (2019). *stringr: Simple, Consistent Wrappers for Common String Operations*. https://CRAN.R-project.org/package=stringr

Wickham, H., François, R., Henry, L., & Müller, K. (2020). *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr

Wickham, H., & Henry, L. (2020). *tidyr: Tidy Messy Data*. https://CRAN.R-project.org/package=tidyr

Wickham, H., & Seidel, D. (2020). *scales: Scale Functions for Visualization*. https://CRAN.R-project.org/package=scales

Wilke, C. O. (2019). *cowplot: Streamlined Plot Theme and Plot Annotations for "ggplot2."* https://CRAN.R-project.org/package=cowplot

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *2*(1–3), 37–52.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., & Funtowicz, M. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, Abs/1910.03771*.

Xie, Y. (2014). knitr: A comprehensive tool for reproducible research in R. *Implementing Reproducible Computational Research*, 3–32.

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). *R markdown: The definitive guide*. CRC Press.

Yarkoni, T., & Westfall, J. (2017). Choosing prediction over explanation in psychology: Lessons from machine learning. *Perspectives on Psychological Science*, *12*(6), 1100–1122.

Appendix I: Examples of Code

```
# Install text from CRAN
install.packages("text")
library(text)

# Set-up en environment with text-required python packages
textrpp_install()

# Initialize the environment - and save the settings for next time
textrpp_initialize(save_profile = TRUE)
```

```
# Look at example data included in the text package comprising both text and numerical variables
(note that there are only 40 participants in this example).
Language_based_assessment_data_8
```

```
# Transform the text/word data to word embeddings (see help(textEmbed) to see the default settings).
word_embeddings <- textEmbed(Language_based_assessment_data_8)

# See how the word embeddings are structured
word_embeddings

# Save the word embeddings to avoid having to embed the text again. It is good practice to save
output from analyses that take a lot of time to compute, which is often the case when analyzing
text data.
saveRDS(word_embeddings, "word_embeddings.rds")

# Get the saved word embeddings (again)
word_embeddings_saved <- readRDS("word_embeddings.rds")
```

```
# Get hidden states for "I am fine"
imf_embeddings_11_12 <- textEmbedLayersOutput("I am fine",
                                              layers = 11:12,
                                              decontexts = FALSE)
imf_embeddings_11_12
```

```
# 1. Concatenate layers(results in 1,536 dimensions).
textEmbedLayerAggregation(imf_embeddings_11_12$context,
                          layers = 11:12,
```

```
                          aggregate_layers = "concatenate",
                          aggregate_tokens = "mean")

# 2. Only select layer 11 (768 dimensions).
textEmbedLayerAggregation(imf_embeddings_11_12$context,
                          layers = 11,
                          aggregate_tokens = "mean")
```

```
# Examine the relationship between satisfactiontext and the corresponding rating scale

model_satisfactiontext_swls <- textTrain(
  x = word_embeddings$satisfactiontexts,  # the predictor variables (i.e., the word embeddings)
  y = Language_based_assessment_data_8$swlstotal, # the criterion variable (i.e., the rating
scale score).
  model_description = "author(s): Kjell, Giorgi, & Schwartz; data: N=40, population = Online,
Mechanical Turk; publication: title = Example for demo; description: swls = the satisfaction with
life scale")

# Examine the correlation between predicted and observed Harmony in life scale scores
model_satisfactiontext_swls$results
```

```
# Examine the names in the object returned from training
names(model_satisfactiontext_swls)
```

```
# Predicting several outcomes from several word embeddings
models_words_ratings <- textTrainLists(word_embeddings[1:2],
Language_based_assessment_data_8[5:6])

# See results
models_words_ratings$results
```

```
# Download and read a valence trained prediction model
# To download the model, go to the following web address: https://r-
text.org/text_models/valence_Warriner_L11_12.rds
# Save the model to your working directory and run:
valence_Warriner_L11_12 <- readRDS("valence_Warriner_L11_12.rds")

# Examine the model
valence_Warriner_L11_12
```

```
# Apply the model to the satisfaction text
satisfaction_text_valence <- textPredict(valence_Warriner_L11_12,
word_embeddings$satisfactiontexts)

# Examine the correlation between the predicted valence and the Satisfaction with life scale
score
psych::corr.test(satisfaction_text_valence$.pred, Language_based_assessment_data_8$swlstotal)
```

```
# Compute semantic similarity scores between two text columns, using the previously created
word_embeddings.
semantic_similarity_scores <- textSimilarity(word_embeddings$harmonytexts,
                                             word_embeddings$satisfactiontexts)
# Look at the first scores
head(semantic_similarity_scores)
```

```
#  Read word norms text  (later we will use these for the semantic centrality plot)
word_norms <- read.csv("Word_Norms_Mental_Health_Kjell2018_text.csv")
```

```
# Read the word embeddings for the word norms
word_norms_embeddings <- readRDS("Word_Norms_Mental_Health_Kjell2018_text_embedding_L11_12.rds")

# Examine which word norms there are.
names(word_norms_embeddings)
```

```
# Compute semantic similarity score between the harmony answers and the harmony norm
# Note that the descriptive word answers are used instead of text answers to correspond with how
the word norm was created.
norm_similarity_scores_harmony <- textSimilarityNorm(word_embeddings$harmonywords,
                                                     word_norms_embeddings$harmonynorm)

# Correlating the semantic measure with the corresponding rating scale
psych::corr.test(norm_similarity_scores_harmony, Language_based_assessment_data_8$hilstotal)
```

```
# Test the semantic difference between the satisfaction with life texts and the harmony in life
text.
s_test <- textSimilarityTest(word_embeddings$satisfactiontexts,
                             word_embeddings$harmonytexts,
```

```
                          method = "paired",
                          Npermutations = 100)

s_test



library(tidyverse)
# Merge satisfaction and harmony words to one column
worddata_merged <- c(Language_based_assessment_data_8$satisfactionwords,
```

```
                    Language_based_assessment_data_8$harmonywords)

# Create variable indicating whether it is a satisfaction or harmony word response
satwor1_harwor2 <- c(rep(1, length(Language_based_assessment_data_8$satisfactionwords)),
                     rep(2, length(Language_based_assessment_data_8$harmonywords)))

# Replicating the SWLS score so that it has equal length as the merged words and satwor1_harwor2
swls_scores <- c(Language_based_assessment_data_8$swlstotal,
                 Language_based_assessment_data_8$swlstotal)

# Merge satisfaction and harmony embeddings
embedding_merged <- bind_cols(word_embeddings$satisfactionwords,
                              word_embeddings$harmonywords)


# Pre-processing data for plotting
projection_results <- textProjection(worddata_merged,
                                     embedding_merged,
                                     word_embeddings$singlewords_we,
                                     satwor1_harwor2,
                                     swls_scores
)
projection_results$word_data


# Supervised Dimension Projection Plot
# To avoid warnings -- and that words do not get plotted, first increase the max.overlaps for the
entire session:
options(ggrepel.max.overlaps = 1000)

plot_projection <- textProjectionPlot(
  word_data = projection_results,
  min_freq_words_plot = 1,
  plot_n_word_extreme = 10,
  plot_n_word_frequency = 5,
  plot_n_words_middle = 5,
  y_axes = FALSE,
  p_alpha = 0.05,
  p_adjust_method = "fdr",
  title_top = "Supervised Dimension Projection",
  x_axes_label = "Satisfaction words versus Harmony words",
  y_axes_label = "",
  bivariate_color_codes = c("#60A1F7", "#5DC688", "#40DD52",
                            "#398CF9", "#EAEAEA", "#85DB8E",
                            "#E07f6a", "#FF0000", "#EA7467")
)
# View plot
plot_projection$final_plot
```

```r
# Supervised Dimension Projection Plot
plot_projection_2D <- textProjectionPlot(
  word_data = projection_results,

  y_axes = TRUE,

  p_adjust_method = "fdr",
  title_top = "Supervised Two-Dimension Projection",
  x_axes_label = "Satisfaction words versus Harmony words",
  y_axes_label = "Satisfaction with life scale score",
```

```
  bivariate_color_codes = c("#60A1F7", "#5DC688", "#40DD52",
                            "#398CF9", "#EAEAEA", "#85DB8E",
                            "#E07f6a", "#FF0000", "#EA7467")
)
# View plot
plot_projection_2D$final_plot
```

```
# Computing words' centrality (semantic similarity) score to the aggregated embedding of all words
centrality_results <- textCentrality(words = word_norms$satisfactionnorm,
                                     word_embeddings = word_norms_embeddings$satisfactionnorm,
                                     word_norms_embeddings$singlewords_we)

centrality_plot <- textCentralityPlot(word_data=centrality_results,
                                      min_freq_words_test = 2,
                                      plot_n_word_extreme = 10,
                                      plot_n_word_frequency = 5,
                                      plot_n_words_middle = 5,
                                      title_top = "Satisfaction with life word norm: Semantic
Centrality Plot",
                                      x_axes_label = "Satisfaction with Life Semantic Centrality")

centrality_plot$final_plot
```