Investigating morphological processing

Experimental and computational approaches

Hands-on sessions

Fritz Günther & Marco Marelli

Spring School Bolzano 2021

Investigating morphological processing with distributional semantics

Structure

- Getting the word vectors

Investigating morphological processing with distributional semantics

Structure

- Getting the word vectors
- Introduction to DISSECT (python toolkit)

Investigating morphological processing with distributional semantics

Structure

- Getting the word vectors
- Introduction to DISSECT (python toolkit)
  - Building the vector space
  - Training the compositional model
  - Applying the compositional model
- Introduction to LSAfun (R package)

Investigating morphological processing with distributional semantics

Structure

- Getting the word vectors
- Introduction to DISSECT (python toolkit)
  - Building the vector space
  - Training the compositional model
  - Applying the compositional model
- Introduction to LSAfun (R package)
  - Computing similarities
  - Exploring neighborhoods

Investigating morphological processing with distributional semantics

Structure

- Getting the word vectors
- Introduction to DISSECT (python toolkit)
  - Building the vector space
  - Training the compositional model
  - Applying the compositional model
- Introduction to LSAfun (R package)
  - Computing similarities
  - Exploring neighborhoods
- A little empirical analysis of behavioral data

Getting the word vectors

Getting the word vectors

Pre-built options

Count- and prediction-based vectors by
Baroni, Dinu, & Kruszewski (2014). *Don't count, predict! A systematic comparison of context-counting vs context-predicting semantics vectors.*

(and other resources)

https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC

Getting the word vectors

Pre-built options

Count- and prediction-based vectors by
Mandera, Keuleers, & Brysbaert (2017). *Explaining human performance in psycholinguistic tasks with models of semantic similarity based on prediction and counting: A review and empirical validation.*

http://meshugga.ugent.be/snaut//spaces/

Getting the word vectors

Pre-built options

GloVe models by
Pennington, Socher, & Manning (2014). *GloVe: Global Vectors for Word Representation.*

https://nlp.stanford.edu/projects/glove/

Getting the word vectors

Pre-built options

fastText models by
Grave, Bojanowski, Gupta, Joulin, & Mikolov (2018). *Learning Word Vectors for 157 Languages.*

https://github.com/facebookresearch/fastText/blob/master/docs/crawl-vectors.md

Getting the word vectors

Pre-built options

BERT models

Turc, Chang, Lee, & Toutanova (2019). *Well-Read Students Learn Better: On the Importance of Pre-training Compact Models.*

https://github.com/google-research/bert

Getting the word vectors

Pre-built options

My own semantic space repository

Günther, Dudschig, & Kaup (2015). *LSAfun – An R package for computations based on Latent Semantic Analysis.*

https://sites.google.com/site/fritzgntr/software-resources/semantic_spaces

Getting the word vectors

Building your own vectors

- You need a (large) corpus

Getting the word vectors

Building your own vectors

- You need a (large) corpus
- This corpus typically needs to be pre-processed in a certain way (e.g., one word per line, or one document per line)

Getting the word vectors

Building your own vectors

- You need a (large) corpus
- This corpus typically needs to be pre-processed in a certain way (e.g., one word per line, or one document per line)

Getting the word vectors

Building your own vectors

Python library: *gensim*

https://radimrehurek.com/gensim/

Getting the word vectors

Building your own vectors

Python library: *DISSECT*

https://github.com/composes-toolkit/dissect/tree/python3

Getting the word vectors

Building your own vectors

R packages: *rword2vec* and *word2vec*

https://github.com/mukul13/rword2vec
https://cran.r-project.org/web/packages/word2vec/

Getting the word vectors

Building your own vectors

TensorFlow

https://www.tensorflow.org/tutorials/text/word_embeddings

The DISSECT toolkit

The DISSECT toolkit

General information

- Python toolkit for working with distributional semantics

The DISSECT toolkit

General information

- Python toolkit for working with distributional semantics
- Building semantic spaces
- Composition
- Similarities

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC
- Includes introduction on python code as well as command-line usage

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC
- Includes introduction on python code as well as command-line usage
- Here, we will focus on command-line usage

The DISSECT toolkit
Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC

The DISSECT toolkit

Documentation and Tutorial

- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC
- Includes introduction on python code as well as command-line usage

The DISSECT toolkit
Documentation and Tutorial

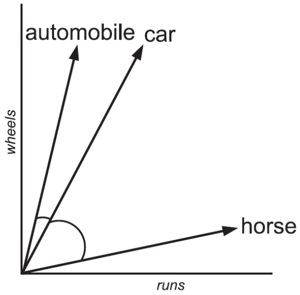
- DISSECT has an excellent documentation and tutorial
- Download available at:
  https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC
- Includes introduction on python code as well as command-line usage
- Here, we will focus on command-line usage

The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

- The (distributional) semantic space contains (distributional) semantic vectors representing word meanings

	runs	*wheels*
automobile	1	4
car	2	4
horse	4	1



The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

- The actual file (dense matrix format, `dm`): One line per vector, word as the first entry, followed by the N dimensional values, no headline
- N needs to be the same for all words
- Example:

```
happy     1.23   -0.12  2.33  - 1.22
unhappy   1.44    1.10  0.02   -1.11
familiar  0.11  - 0.22  2.94   -1.35
```

The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

The `build_core_space.py` function

```
python build_core_space.py [options] [config_file]
```

The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

The `build_core_space.py` function

```
python build_core_space.py [options] [config_file]
```

The options are:

`-i, --input` Prefix of the input files.

`--input_format` Input format of the file containing co-occurrence counts: one of sm (sparse matrix), dm (dense matrix), pkl (pickle), see *information about the input formats*.

`-o, --output` Output directory. For each specification of space creation parameters, a file named CORE_SS.inputname.parameters.format will be left in this directory.

Example:

```
python build_core_space.py -i ../examples/data/in/ex01
--input_format sm -o ../examples/data/out/
```

The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

For more options, see the Documentation!

The DISSECT toolkit

Step 1: Building the space (Documentation: /toolkit/creating.html)

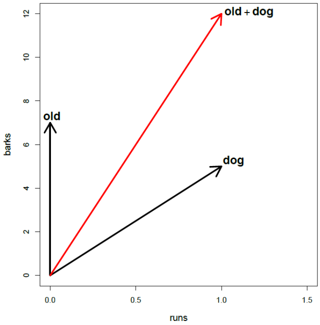
Now you!

- Use `build_core_space.py` to build a space from the file `baroni.dm` (in the Materials for this course)
- In addition to other output, this will always produce the .pkl file we need to continue

The DISSECT toolkit

Step 2: Training a composition model

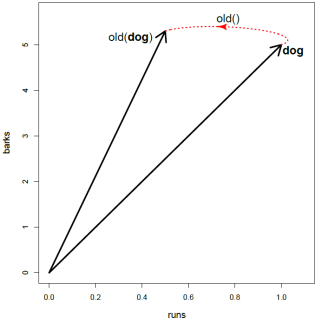
- Mixture-based models (such as the Additive Model): Arithmetic operation on constituent vectors
- Both constituents need to have vector representations in the semantic space



The DISSECT toolkit

Step 2: Training a composition model

- Lexical Functions (such as FRACSS): One constituent is a function mapping the other constituent onto the combined meaning
- The function does not necessarily need a vector representation in the semantic space

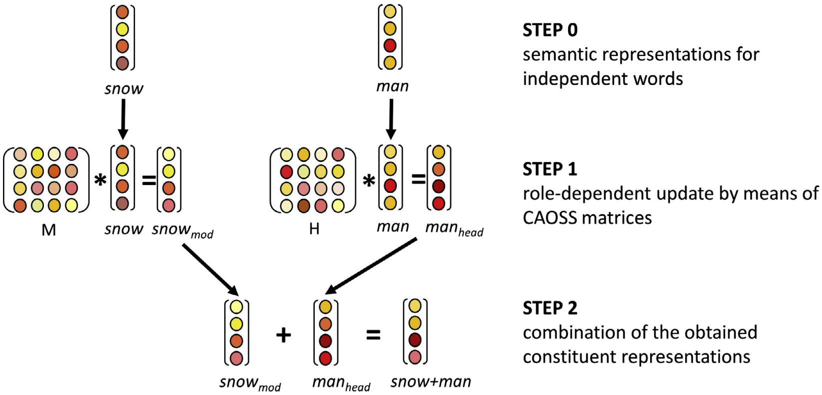


for our purpose, replace *old* with *un-* and *dog* with *happy*

The DISSECT toolkit

Step 2: Training a composition model

- CAOSS model: Combination of (1.) functional mapping and (2.) mixture (addition)



The DISSECT toolkit

Step 2: Training a composition model

- In our course, we focus on a small example:
  A FRACSS model just for the prefix **un-**

The DISSECT toolkit

Step 2: Training a composition model

- In our course, we focus on a small example:
  A FRACSS model just for the prefix **un-**

- First, we need to identify a training set
  This set consists of words with the prefix un- and their stems

The DISSECT toolkit

Step 2: Training a composition model

- In our course, we focus on a small example:
  A FRACSS model just for the prefix **un-**

- First, we need to identify a training set
  This set consists of words with the prefix un- and their stems

- For each pair, both the complex word and the stem need to have a vector in the semantic space

The DISSECT toolkit

Step 2: Training a composition model

- In our course, we focus on a small example:
  A FRACSS model just for the prefix **un-**

- First, we need to identify a training set
  This set consists of words with the prefix un- and their stems

- For each pair, both the complete word and the stem need to have a vector in the semantic space

- More specifically, the file including the training set needs to look like this:

```
un-   happy      unhappy
un-   fair       unfair
un-   grateful   ungrateful
...   ...        ...
```

The DISSECT toolkit

Step 2: Training a composition model

Now you!

- The file `baroni.rows` (in the Materials for this course) contains all words available in the semantic space
- Use this file to identify a training set, using a program and method of your choice, saving it as `UN_trainset.txt`
- For now, just focus on pairs *(un[stem],[stem])* for which both *un[stem]* and *[stem]* are available in the file `baroni.rows`
- Remember, the final file needs to look like this:

```
un-   happy      unhappy
un-   fair       unfair
un-   grateful   ungrateful
...   ...        ...
```

The DISSECT toolkit

Step 2: Training a composition model

Now you!

- Inspect the file

The DISSECT toolkit

Step 2: Training a composition model

Now you!

- Inspect the file
- What are some potential problems of our selection procedure?
- How can we avoid these problems?

The DISSECT toolkit

Step 2: Training a composition model

Now you!

- Inspect the file
- What are some potential problems of our selection procedure?
- How can we avoid these problems?
- $\rightarrow$ For example, use annotated resources such as CELEX

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

The `train_composition.py` function

```
python train_composition.py [options] [config_file]
```

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

The `train_composition.py` function

```
python train_composition.py [options] [config_file]
```

The options are:

`-i, --input` Input file containing a list of element1 element2 phrase tuples on each line. The words (or phrases) in columns 1 and 2 will be extracted from the argument space, the phrase in column 3 from the phrase space. When training a Lexical Function model, the first column (element1) will contain a functor name, and the element2 and phrase vectors will be used as an input-output training pair when estimating the corresponding function (a separate function will be trained for each distinct element1 in the file).

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

`-o, --output` Output directory of the resulting composition model. The output is a pickle dump of the composition model object, named TRAINED_COMP_MODEL.model_name.input_file.pkl, e.g., TRAINED_COMP_MODEL.weighted_add.mytrainingfile.pkl.

`-m, --model` Name of a composition model to be trained. One of weighted_add (Weighted Additive), full_add (Full Additive), lexical_func (Lexical Function) or dilation (Dilation).

`--export_params: True/False` If True, parameters of the learned model are exported to an appropriate format. Optional, False by default.

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

`-a, --arg_space` File containing the space of the arguments (i.e., element1 and element2). Pickle format (and .pkl extension) required.

`-p, --phrase_space` File containing the phrase space (i.e., the space that contains the phrase part of the element1 element2 phrase tuples) used for training. Pickle format (and .pkl extension) required.

- When working with morphologically complex words, the argument space and the peripheral space are identical

Example:

```
python train_composition.py
-i ../examples/data/in/train_data.txt -m lexical_func
-a ../examples/data/out/ex01.pkl
-p ../examples/data/out/PHRASE_SS.ex10.pkl
-o ../examples/data/out/ --export_params True
```

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

For more options, see the Documentation!

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

Now you!

- Use `train_composition.py` to train a Lexical Function model with the file containing our training set from the previous step
- *un-* is the Lexical Function mapping *[stem]* onto *un[stem]*

The DISSECT toolkit

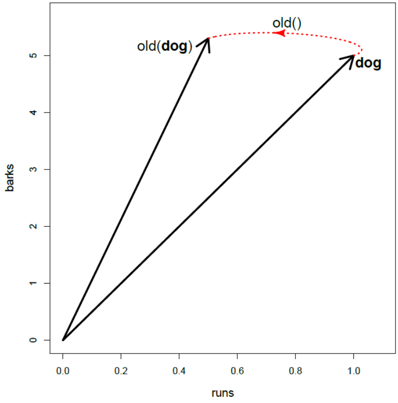
Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- Once the composition model is trained, you can apply it to any vector to create compositional vectors

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- Once the composition model is trained, you can apply it to any vector to create compositional vectors



The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- For example, apply the Lexical Function for **un-** to *silly* to create a vector for *unsilly*

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- For example, apply the Lexical Function for **un-** to *silly* to create a vector for *unsilly*
- Can be done for novel combinations (unsilly), but also for existing words: Apply **un-** to *happy* to create a compositional vector for *unhappy*

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- For example, apply the Lexical Function for **un-** to *silly* to create a vector for *unsilly*
- Can be done for novel combinations (unsilly), but also for existing words: Apply **un-** to *happy* to create a compositional vector for *unhappy*
- Think of these compositional vectors for familiar compounds as "Which meaning would someone expect who doesn't know the lexicalized meaning of the word"

The DISSECT toolkit

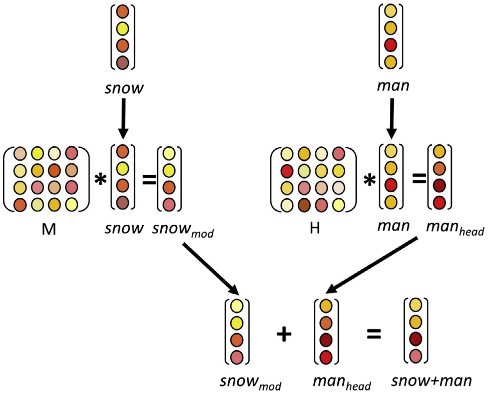
Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- The same for other compositional models:

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- The same for other compositional models:
- Construct a compositional compound vector from its two constituent vectors using the CAOSS model



STEP 0
semantic representations for independent words

STEP 1
role-dependent update by means of CAOSS matrices

STEP 2
combination of the obtained constituent representations

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

The `apply_composition.py` function

```
python apply_composition.py [options] [config_file]
```

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

The `apply_composition.py` function

```
python apply_composition.py [options] [config_file]
```

The options are:

`-i, --input` Input file containing a list of element1 element2 composed_phrase tuples on each line. The words (or phrases) in column 1 will be composed with the words (or phrases) in column 2. A semantic space for the composed words is created using the strings in column 3 as phrase labels (note that the latter strings are arbitrary, they have no mandatory relation to word1 and word2). If the Lexical Function model is applied, element1 is interpreted as the name of the functor to be used, element2 as the argument.

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- This means that the input is supposed to look like this:

```
un-   happy   unhappy
un-   silly   unsilly
...   ...     ...
```

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- This means that the input is supposed to look like this:

```
un-   happy   unhappy
un-   silly   unsilly
...   ...     ...
```

- But it can also look like this:

```
un-   happy   unhappy__cmp
un-   silly   unsilly__cmp
...   ...     ...
```

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

- This means that the input is supposed to look like this:

```
un-   happy   unhappy
un-   silly   unsilly
...   ...     ...
```

- But it can also look like this:

```
un-   happy   unhappy__cmp
un-   silly   unsilly__cmp
...   ...     ...
```

- In principle, nothing stops you from doing this:

```
un-   happy   dragonman
un-   silly   fhjd444dfF
...   ...     ...
```

The DISSECT toolkit

Step 2: Training a composition model (Documentation: /toolkit/composing.html)

Now you!

- Create a new file `UN_applset.txt` from the training set in `UN_trainset.txt`, transforming it from this:

```
un-   happy      unhappy
un-   fair       unfair
un-   grateful   ungrateful
...   ...        ...
```

  to this:

```
un-   happy      unhappy__cmp
un-   fair       unfair__cmp
un-   grateful   ungrateful__cmp
...   ...        ...
```

  This will allow us to easily distinguish *observed* from *compositional* vectors later on, which is very useful

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

`-o, --output` Output directory of the resulting composed space. The output is a pickle dump of the composed space (and possibly a sparse or dense file with the same data if requested with –output_format option). The output files are named COMPOSED_SS.model_name.input_file.format, e.g., COMPOSED_SS.Dilation.myphrases.txt.pkl.

`--output_format: additional_output_format` Additional output format for the resulting composed space: one of sm (sparse matrix), dm (dense matrix). This is in addition to default pickle output. Optional.

`-a, --arg_space` File(s) containing the space(s) of the arguments. If a second file is provided, the second element of a pair is interpreted in the additional space. Pickle format (and .pkl extension) required.

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

`-m, --model` Name of the composition model to be applied, whose parameters will be directly specified on the command line (instead of being read from model file). One of mult (Multiplicative), weighted_add (Weighted Additive) or dilation (Dilation) is expected. One (and only one) of -m or –load_model has to be provided.

`--load_model model_file` File containing a previously saved composition model (pickle dump). One (and only one) of -m or –load_model has to be provided.

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

Example:

```
python apply_composition.py
-i ../examples/data/in/data_to_comp.txt
--load_model ../examples/data/out/model01.pkl
-a ../examples/data/out/ex01.pkl -o ../examples/data/out/
--output_format dm
```

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

Now you!

- Use `apply_composition.py` to apply the Lexical Function model trained in the previous step onto the stems in `UN_applset.txt`, in order to create vectors for the `unhappy__cmp`-style expressions in this file
- Inspect the resulting `COMPOSED_SS` file
- Repeat the same process for the novel combinations in the file `UN_novelwords.txt` (in the Materials for this course)

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

Good job!

The DISSECT toolkit

Step 3: Applying a composition model (Documentation: /toolkit/composing.html)

Good job!

We now have everything we need to proceed!

The DISSECT toolkit

FRACSS: Further notes

- If you want to train FRACSS for more than one affix, all you need to do is extending the files containing the training (and application) sets:

```
un-   happy   unhappy
un-   fair    unfair
...   ...     ...
mis-  cast    miscast
mis-  match   mismatch
...   ...     ...
-ist  violin  violinist
-ist  guitar  guitarist
...   ...     ...
```

The DISSECT toolkit

Composition models: Further notes

- Note that *you* decide what counts as a training item

The DISSECT toolkit

Composition models: Further notes

- Note that *you* decide what counts as a training item
- Words don't need to be separable at the surface level

`-ist`	`cycle`	`cyclist`
`-ness`	`happy`	`happiness`

The DISSECT toolkit

Composition models: Further notes

- Note that *you* decide what counts as a training item
- Words don't need to be separable at the surface level

`-ist`	`cycle`	`cyclist`
`-ness`	`happy`	`happiness`

- Words don't necessarily need to be transparent or etymologically related

`-less`	`fruit`	`fruitless`
`-er`	`corn`	`corner`
`un-`	`ion`	`union`

The DISSECT toolkit

Composition models: Further notes

- Note that *you* decide what counts as a training item
- Words don't need to be separable at the surface level

`-ist`	`cycle`	`cyclist`
`-ness`	`happy`	`happiness`

- Words don't necessarily need to be transparent or etymologically related

`-less`	`fruit`	`fruitless`
`-er`	`corn`	`corner`
`un-`	`ion`	`union`

- In principle, nothing stops you from inserting complete nonsense

`-less`	`karma`	`chameleon`
`-derp`	`door`	`universe`

The DISSECT toolkit

Composition models: Further notes

How can you tell the difference between a chemist and a plumber?

The DISSECT toolkit

Composition models: Further notes

How can you tell the difference between a chemist and a plumber?

Ask them to pronounce "unionized"

The DISSECT toolkit

Compound words: The CAOSS model

- For compound words in the CAOSS model, the process is the same as described for the FRACSS model, with three differences:

The DISSECT toolkit

Compound words: The CAOSS model

- For compound words in the CAOSS model, the process is the same as described for the FRACSS model, with three differences:
  - For the `train_composition.py` function, use `-m full_add` instead of `-m lexical_func`

The DISSECT toolkit

Compound words: The CAOSS model

- For compound words in the CAOSS model, the process is the same as described for the FRACSS model, with three differences:
  - For the `train_composition.py` function, use `-m full_add` instead of `-m lexical_func`
  - The training set (containing all compounds and their constituents) looks as follows, and *all* entries (also the first column) need to be entries in the semantic space

```
sun      rise         sunrise
singer   songwriter   singer-songwriter
```

The DISSECT toolkit

Compound words: The CAOSS model

- For compound words in the CAOSS model, the process is the same as described for the FRACSS model, with three differences:
  - For the `train_composition.py` function, use `-m full_add` instead of `-m lexical_func`
  - The training set (containing all compounds and their constituents) looks as follows, and *all* entries (also the first column) need to be entries in the semantic space

```
sun      rise         sunrise
singer   songwriter   singer-songwriter
```

  - The application set looks as follows, and all entries in the first two columns need to be entries in the semantic space

```
sun      rise   sunrise__cmp
monkey   ring   monkeyring__cmp
```

The DISSECT toolkit

Further functionalities

- Using the `compute_similarities.py` function, you can already compute similarities between vectors in DISSECT
- For more functions and integration with empirical data analyses, we will move to R and the *LSAfun* package to perform this step

The LSAfun package

The LSAfun package

General Information

- Created during my PhD
- Name: *LSA* (Latent Semantic Analysis) is an early distributional model; *fun* for functions
- For *computations on* semantic spaces, but not *creation of* semantic spaces
- Includes some useful functionalities for working with semantic spaces

The LSAfun package

General Information

- For a complete tutorial, see

  Günther, F., Dudschig, C., & Kaup, B. (2015). LSAfun – An R package for computations based on Latent Semantic Analysis. *Behavior Research Methods, 47,* 930-944.

- Core functionalities:
  - **Computing similarities**
  - **Neighborhood computations**
  - **Plots and multidimensional scaling**
  - Other applied functions

The LSAfun package

Starting your R session

1. Open R or RStudio
2. Open a script or create a new script
3. Set the working directory to the most convenient path, such as:
   `setwd("G:/Lehre/Spring School Bolzano 2021/"))`

The LSAfun package

First step: Loading a semantic space

Loading a semantic space

- From plain text (example: space- or tab-separated file):

  ```
  myspace <-
  as.matrix(read.table("file.txt",row.names = 1))
  ```

  - This can take quite a bit of time depending on the size of the file

- From R's .rda format (as on https://sites.google.com/site/fritzgntr/software-resources/semantic_spaces):

  ```
  load("filename.rda")
  ```

  - This is pretty fast also for larger spaces
  - Loads an R object already with a variable name

The LSAfun package

First step: Loading a semantic space

Now you!

- Open the `R_script_bolzano_STUDENTS.R` file in R and set a useful working directory
- The core semantic space is already saved as `baroni.rda` (available in the Materials for this course). Load it using the `load()` function.
- Load your compositional spaces in the dense matrix format (named `COMPOSED...dm`) using the `read.table()` function.
- Use the `rbind()` function to combine all spaces into one big space named `myspace`:,

  ```
  myspace <- rbind(space1,space2,space3)
  ```

  Note: Make sure that no row names are duplicated

The LSAfun package

First step: Loading a semantic space

Inspecting the space

`is(myspace)`	what kind of object is it?
`str(myspace)`	the structure of the object
`nrow(myspace)`	number of rows
`ncol(myspace)`	number of columns
`dim(myspace)`	dimensionality of the matrix
`rownames(myspace)`	the row names
`head(rownames(myspace))`	only the first row names
`head(myspace)`	the first rows of the matrix
`any(duplicated(rownames(myspace)))`	any duplicated row names?

The LSAfun package

Computing similarities

Core function:

```
Cosine("word1","word2",tvectors = myspace)
```

- `tvectors` defines the semantic space in which the similarity is computed
  (needs to be a numerical matrix)
- `word1` and `word2` need to be entries of the semantic space
  (more specifically, they need to be elements of
  `rownames(myspace)`)

The LSAfun package

Computing similarities

```
multicos("word1 word2 word3",tvectors = myspace)
```

```
multicos("word1 word2 word3","word4 word5", tvectors
= myspace)
```

- Input can be of format `"word1 word2 word3"` or `c("word1", "word2", "word3")`
- Input can also consist of single words
- Computes a cosine matrix including all pairwise similarities
- If no second argument is provided, the first argument will automatically be repeated as the second

The LSAfun package

Computing similarities

Now you!

- Compute the semantic transparency (in relatedness terms) of *unhappy*: Cosine similarity between *happy* and the observed vector for *unhappy*
- Compute the semantic transparency (in compositional terms) of *unhappy*': Cosine similarity between *happy* and the compositional vector for *unhappy*
- Compute the compositionality (meaning predictability) of *unhappy*: Cosine similarity between the observed and compositional vector for *unhappy*
- Repeat for *union*

The LSAfun package

Computing similarities: Additional functions

```
costring("word1 word2 word3","word4 word5",tvectors =
myspace)
```

- Input can be of format `"word1 word2 word3"` or `c("word1", "word2", "word3")`
- Input can also consist of single words
- Computes the cosine between the two "sentences/ documents"
- Vectors for "sentences/ documents" defined as vector sum of the individual words

The LSAfun package

Computing similarities: Additional functions

```
multicostring("word1 word2 word3","word4
word5",tvectors = myspace)
```

- Input can be of format `"word1 word2 word3"` or `c("word1", "word2", "word3")`
- Input can also consist of single words
- Computes the cosine betweens the “sentence/ document” in the first argument and all the words in the second argument

The LSAfun package

Computing similarities: Additional functions

```
pairwise(c("word1", "word2", "word3"),c("word4",
"word5", "word6"),tvectors = myspace)
```

- Computes pairwise similarities between the first elements in the two vectors, the second elements in the two vectors, and so on (here: word1–word4, word2–word5, word4–word6)
- Useful when working with lists of words in a dataframe (the most common data type in R)

The LSAfun package

First step: Loading a semantic space

Now you!

- Run the few lines of code directly under `## create data frame with affixed words and stems` – how does the resulting object `dat` look like?
- Compute the compositionality (similarity between observed and compositional vectors for a complex words) for all complex words in `dat` and store the result as a new column in `dat`
  - To access an individual column of a dataframe such as `dat`, use for example `dat$Word`
  - You can use `dat$newvar <- VALUE` to create a new colum in `dat`

The LSAfun package

First step: Loading a semantic space

Now you!

- Repeat for semantic transparency (similarity between stem and complex word), both for the relatedness version and the compositional version of semantic transparency
- Compute the correlation between the three variables, using `cor(dat$varname1,dat$varname2)`
  At the same time, have a look at these relations:
  `plot(dat$varname1,dat$varname2)`

The LSAfun package

Exploring neighborhoods

- At first sight, distributional vectors are somewhat opaque:
  - What do these numbers mean?
  - How do I know if my model does anything sensible?
- We already looked at a good option: Calculating similarities to other words
  - This is especially straightforward for complex words (which have a stem) and compositional vectors (which can have an observed counterpart)

The LSAfun package

Exploring neighborhoods

- We now use these similarities to explore neighborhoods
- n nearest neighbors of a word $=$ n words with the highest cosine similarity to that word

The LSAfun package

Exploring neighborhoods

```
neighbors("word", n = 50, tvectors = myspace)
```

- Define the word, the number of neighbors, and the semantic space you want to search in
- Can take a bit of time depending on the size of the semantic space: Needs to calculate cosine similarities between the word and all other words in the semantic space

The LSAfun package

Exploring neighborhoods

Now you!

- Select a word with a high compositionality score (*uninstall*), and compute the 50 nearest neighbors of its observed vector
  - In the `baroni` space of observed vectors only
  - In the combined space with all observed and compositional vectors
- Repeat the same for the *compositional* vector of (*uninstall*)
  - In the combined space with all observed and compositional vectors

The LSAfun package

Exploring neighborhoods in a graph

```
plot_neighbors("word", n = 50, tvectors = myspace)
```

- In principle, same syntax as the `neighbors()` function
- Projection of the high-dimensional neighborhood onto a low-dimensional space (2D plane or 3D space)

The LSAfun package

Exploring neighborhoods in a graph

Further optional arguments (see `?plot_neighbors`):

- `dims`: Dimensionality of the plot
- `connect.lines`: How many lines connecting each word to other words
- `start.lines`: Draw lines from the word whose neighborhood is displayed?
- `cex`: Size of words in the plot
- `alpha`: Luminance of the lines
- `alpha.grade`: Proportionally scaling the luminance of the lines
- `col`: Color of the lines

The LSAfun package

Exploring neighborhoods in a graph

Now you!

- Again, for a word with a high compositionality score (*uninstall*), plot the 50 nearest neighbors of its observed vector in the combined space with all observed and compositional vectors (using a 3D plot)
- Play around with some options
- Repeat for the word's compositional vector
- Repeat both for a word with a low compositionality score (*unfabulous*)
- Repeat for the compositional vector of a novel word – does the output seem sensible?

The LSAfun package

Empirical analyses

- Looking at similarities and neighborhoods is nice, but not a systematic investigation

The LSAfun package
Empirical analyses

- Looking at similarities and neighborhoods is nice, but not a systematic investigation
- When investigating morphological representation and processing, we want to compare our model predictions against actual empirical data

The LSAfun package

Empirical analyses

- Looking at similarities and neighborhoods is nice, but not a systematic investigation
- When investigating morphological representation and processing, we want to compare our model predictions against actual empirical data
- Two purposes:
    1. Evaluate the model: Does it make sense?
    2. When evaluated, use the model to investigate empirical questions

The LSAfun package

Empirical analyses

This is not a statistics or data analysis class, so we will focus on very simple examples:

- Correlation between semantic transparency/ compositionality and processing times
- Correlation between semantic transparency/ compositionality and ratings

The LSAfun package

Empirical analyses: Basic steps

Reading a dataset in R

- Use any of the generic `read` functions in R, such as
  - `read.table()` for plain text
  - `read.csv() or read.csv()` for .csv files
- If the first line of the document contains the variable names (usually the case), use the argument `header = T`; otherwise, use `header = F`
- If you are unsure about the functions, arguments, and options, you can always use *Import Dataset* in RStudio

The LSAfun package

Empirical analyses: Basic steps

Merging datasets

- You often need to combine several separate datasets into one: Assume that one dataset contains words and their response times, and the other contains words and their semantic transparency scores
- The `merge()` function in R: `merge(dat1,dat2)`
- Will identify identical column names in `dat1` and `dat2`, look for common entries, and merge the files at these common entries

The LSAfun package

Empirical analyses: Basic steps

Merging datasets: Examples

```
> dat1
     names age
1    Carlo  24
2     Luca  29
3    Mario  33
4 Cristina  17
```

```
> dat2
     names residence
1    Carlo    Milano
2     Luca   Bolzano
3  Adriano   Bologna
4 Cristina   Trieste
```

```
> merge(dat1,dat2)
     names age residence
1    Carlo  24    Milano
2 Cristina  17   Trieste
3     Luca  29   Bolzano
```

```
> merge(dat1,dat2,all.x=T)
     names age residence
1    Carlo  24    Milano
2 Cristina  17   Trieste
3     Luca  29   Bolzano
4    Mario  33      <NA>
```

```
> merge(dat1,dat2,all.x=T)
     names age residence
1    Carlo  24    Milano
2 Cristina  17   Trieste
3     Luca  29   Bolzano
4    Mario  33      <NA>
```

```
> merge(dat1,dat2,all.x=T,all.y=T)
     names age residence
1  Adriano  NA   Bologna
2    Carlo  24    Milano
3 Cristina  17   Trieste
4     Luca  29   Bolzano
5    Mario  33      <NA>
```

The LSAfun package

Empirical analyses: Basic steps

Now you!

- Read the file `derived_words_ST.txt` (in the Materials for this course) and save it as `fracss`
  (Source: Marelli, M., & Baroni, M. (2015). Affixation in semantic space: Modeling morpheme meanings with compositional distributional semantics. *Psychological Review, 122(3)*, 485–515.)
- Make sure that the column names in `fracss` and the `dat` object containing our distributional measures (created in the previous step) can be matched (using `head()`, `names()`, or `colnames()`)
- Merge `fracss` and `dat`

The LSAfun package

Empirical analyses: Basic steps

Now you!

- Compute the correlation and plot the relation between ST ratings and
  - our model's ST (relatedness version)
  - our model's ST (compositional version)
  - our model's compositionality

The LSAfun package

Empirical analyses: Basic steps

Now you!

- Read the file `ELP_data.csv` (in the Materials for this course), save it as `elp`, and merge it with `dat`
  (Source: Balota, D.A., Yap, M.J., Cortese, M.J., Hutchison, K.A., Kessler, B., Loftis, B., Neely, J.H., Nelson, D.L., Simpson, G.B., & Treiman, R. (2007). The English Lexicon Project. *Behavior Research Methods, 39,* 445-459.)
- Create a new column named `logRT` that contains log-transformed Lexical Decision Times (apply the `log()` function to column `I_Mean_RT`)
- Compute the correlation and plot the relation between this log-transformed LDT and
  - our model's ST (relatedness version)
  - our model's ST (compositional version)