

Package: wisclabmisc (via r-universe)

October 14, 2026

Type Package

Title Tools to Support the 'WiscLab'

Version 0.1.1.9000

Description A collection of 'R' functions for use (and re-use) across 'WiscLab' projects.

License MIT + file LICENSE

URL <https://github.com/wisclab/wisclabmisc>,
<https://wisclab.github.io/wisclabmisc/>

BugReports <https://github.com/wisclab/wisclabmisc/issues>

Depends R (>= 4.2)

Imports cli, dplyr, fs, gamlss (>= 5.2.0), lifecycle, magrittr,
ordinal, pROC, purrr, rlang (>= 1.1.0), rsample, sessioninfo,
stringr, tibble, tidyr, tidyselect, usethis

Suggests DBI, duckdb, gamlss.dist, ggplot2, knitr, logitnorm, nlme,
numDeriv, rmarkdown, roxygen2, rstanarm, testthat, tidyverse,
withr, xxhashlite

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website brms, rstan, rmarkdown

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake git make libgit2-dev libicu-dev libuv1-dev libssl-dev libx11-dev

Repository <https://tjmahr.r-universe.dev>

Date/Publication 2026-10-14 19:41:34 UTC

RemoteUrl <https://github.com/tjmahr/wisclabmisc>

RemoteRef HEAD

RemoteSha a4a1e8dc93742eed514828c91c6369f89516153f

Contents

audit_wrap	2
brms_args_create	4
check_model_centiles	5
chrono_age	6
compute_empirical_roc	7
compute_overlap_rate	9
compute_predictive_value_from_rates	10
compute_sens_spec_from_ecdf	11
compute_smooth_density_roc	13
data_acq_consonants	14
data_example_intelligibility_by_length	23
data_fake_intelligibility	24
data_fake_rates	25
data_features_consonants	25
data_tocs_items	29
dir_sync_down	32
extract_function_source	34
file_replace_name	36
fit_beta_gamlss	37
fit_gen_gamma_gamlss	41
fit_kmeans	44
format_year_month_age	46
impute_values_by_length	47
info_surprisal	50
join_to_split	51
logitnorm_mean	52
mem_gamlss	53
predict_centiles	53
print_duckdb	54
skip_block	55
tbl_bind	56
tocs_item	57
trapezoid_auc	58
weight_lengths_with_ordinal_model	59

Index

61

Description

An `audit` is a `list()` with two fields:

1. `data`: most recently updated (or poked) data
2. `notes`: a history of the `fn` calls and results applied to the object.

Usage:

- `audit_wrap(data)` wraps some data inside of an audit.
- `audit_peek(x, fn, ...)` applies a function to the data, log the results in the `notes`, but *not* overwrite the audit data.
- `audit_poke(x, fn, ...)` applies a function to the data, log the results in the `notes`, and overwrite the audit data.
- `audit_unwrap()` returns the underlying data.

Usage

```
audit_wrap(data)
```

```
audit_peek(x, fn, ...)
```

```
audit_poke(x, fn, ...)
```

```
audit_unwrap(x)
```

Arguments

<code>data</code>	data to wrap into an audit
<code>x</code>	an audit object
<code>fn</code>	a function to apply to the data inside of the audit object
<code>...</code>	further arguments for <code>fn</code>

Details

There is no `audit_notes()` function at this time because it is not clear what the most convenient or tidy format would be for the notes.

This container was designed for piping a vector of filenames through a series of data validation functions. Therefore, the intended use case is a vector of a data and not larger objects.

Value

`audit_wrap()`, `audit_peek()`, `audit_poke()` return audit objects. `audit_unwrap()` returns the `$data` from an audit.

Examples

```
x <- letters |>
  audit_wrap() |>
  audit_peek(rev) |>
  audit_peek(toupper)

# Note that none of the underlying data has changed. This is useful for
# finding values that fail validation checks.
x

# Use _poke() to update the data:
x <- letters |>
  audit_wrap() |>
  audit_peek(rev) |>
  audit_poke(toupper) |>
  audit_peek(getElement, 1) |>
  audit_peek(function(x) character(0))
x

# Index into the notes to pull the data
x$notes[[3]][["result"]]
```

brms_args_create	<i>Set default arguments for brms model fitting</i>
------------------	---

Description

Set default arguments for brms model fitting

Usage

```
brms_args_create(
  ...,
  .backend = "cmdstanr",
  .threads = 2,
  .chains = 4,
  .cores = 4,
  .iter = 2000,
  .silent = 0,
  .file_refit = "on_change",
  .refresh = 25,
  .control = list()
)
```

Arguments

... arguments to `brms::brm()` to use as default values.

```
.backend, .threads, .chains, .cores, .iter, .silent, .file_refit,
.refresh, .control
```

These arguments set *default* default values. Overwrite these defaults by using the argument without the `.` prefix.

Details

One can set the default value for `adapt_delta` directly (`adapt_delta = .98`). This value will be propagated to `control$adapt_delta`.

Value

A function for generating a list of arguments to `brms::brm()`.

Examples

```
brm_args <- brms_args_create()

# using package-provided defaults
brm_args()

# overwriting a default value
brm_args(iter = 500)

# adapt_delta is handled specially
brm_args(adapt_delta = .95)
# adapt_delta is handled specially
brm_args(adapt_delta = .95)

# We can overwrite the package-provided defaults
other_brm_args <- brms_args_create(iter = 4000, backend = "rstan")
other_brm_args()

# And overwrite them too
other_brm_args(backend = "cmdstanr")
```

`check_model_centiles` *Compute the percentage of points under each centile line*

Description

`check_model_centiles()` computes centiles from a model and computes the calibration of each centile. `check_computed_centiles()` works on a dataframe with precomputed `c[XX]` columns of centiles.

Usage

```
check_model_centiles(
  data,
  model,
  var_x,
  var_y,
  centiles = c(5, 10, 25, 50, 75, 90, 95)
)

check_computed_centiles(data, var_y)
```

Arguments

data	a dataset used to fit a model. If the dataframe is grouped with <code>dplyr::group_by()</code> , sample centiles are computed for each group.
model	a gamlss model prepared by <code>mem_gamlss()</code>
var_x, var_y	bare column names of the predictor and outcome variables
centiles	centiles to use for prediction. Defaults to <code>c(5, 10, 25, 50, 75, 90, 95)</code> .

Value

a tibble the number of points and the percentage of points less than or equal to each quantile value.

chrono_age	<i>Compute chronological age in months</i>
-------------------	--

Description

Ages are rounded down to the nearest month. A difference of 20 months, 29 days is interpreted as 20 months.

Usage

```
chrono_age(t1, t2)
```

Arguments

t1, t2	dates in "yyyy-mm-dd" format
---------------	------------------------------

Value

the chronological ages in months. NA is returned if the age cannot be computed.

Examples

```
# Two years exactly
chrono_age("2014-01-20", "2012-01-20")
#> 24

# Shift a year
chrono_age("2014-01-20", "2013-01-20")
#> 12
chrono_age("2014-01-20", "2011-01-20")
#> 36

# Shift a month
chrono_age("2014-01-20", "2012-02-20")
#> 23
chrono_age("2014-01-20", "2011-12-20")
#> 25

# 3 months exactly
chrono_age("2014-05-10", "2014-02-10")
#> 3

# Borrow a month when the earlier date has a later day
chrono_age("2014-05-10", "2014-02-11")
#> 2, equal to 2 months, 29 days rounded down to nearest month

# Inverted argument order
chrono_age("2012-01-20", "2014-01-20")
#> 24

# Multiple dates
t1 <- c("2012-01-20", "2014-02-10", "2010-10-10")
t2 <- c("2014-01-20", "2014-05-10", "2014-11-10")
chrono_age(t1, t2)
#> [1] 24  3 49
```

`compute_empirical_roc`*Create an ROC curve from observed data*

Description

Create an ROC curve from observed data

Usage

```
compute_empirical_roc(
  data,
  response,
  predictor,
```

```

    direction = "auto",
    best_weights = c(1, 0.5),
    levels = NULL,
    ...
  )

```

Arguments

data	a dataframe containing responses (groupings) and predictor variable
response	a bare column name with the group status (control vs. cases). If the response has more than two groups, the first element of levels is the control group and the second element of levels is the case group.
predictor	a bare column name with the predictor to use for classification
direction	direction for computing case status. <code>pROC::roc()</code> 's direction conventions are supported: "<" (i.e., <code>control < case</code>) and ">" (<code>control > case</code>), or "auto" to have pROC guess the direction (if applicable). Alternatively, more verbose directions are supported: "case-low" or "control-high" if a low score predicts case status (<code>case <= threshold < control</code> , analogous to pROC's ">") and "case-high" or "control-low" if a high score predicts case status (<code>control < threshold <= case</code> , analogous to pROC's "<"). These directions are translated into the pROC conventions.
best_weights	weights for computing the best ROC curve points. Defaults to <code>c(1, .5)</code> , which are the defaults used by <code>pROC::coords()</code> .
levels	two-element vector <code>c(control, case)</code> where control is the value of response for the control group and case is the value of response for the case group. The ordering matters: The first element of the vector names the control group.
...	additional arguments passed to <code>pROC::roc()</code> .

Value

a new dataframe of ROC coordinates is returned with columns for the predictor variable, `.sensitivities`, `.specificities`, `.auc`, `.direction`, `.controls`, `.cases`, `.n_controls`, `.n_cases`, `.is_best_youden` and `.is_best_closest_topleft`.

Examples

```

set.seed(100)
x1 <- rnorm(100, 4, 1)
x2 <- rnorm(100, 2, .5)
both <- c(x1, x2)
steps <- seq(min(both), max(both), length.out = 200)
d1 <- dnorm(steps, mean(x1), sd(x1))
d2 <- dnorm(steps, mean(x2), sd(x2))
data <- tibble::tibble(
  y = steps,
  d1 = d1,
  d2 = d2,
  outcome = rbinom(200, 1, prob = 1 - (d1 / (d1 + d2))),

```

```

    group = ifelse(outcome, "case", "control")
  )

  # get an ROC on the fake data
  compute_empirical_roc(data, outcome, y)
  # this guess the cases and controls from the group name and gets it wrong
  compute_empirical_roc(data, group, y)
  # better
  compute_empirical_roc(data, group, y, levels = c("control", "case"))

```

compute_overlap_rate *Compute overlap rate for (phoneme alignment) intervals*

Description

Compute overlap rate for (phoneme alignment) intervals

Usage

```
compute_overlap_rate(x1, x2, y1, y2)
```

Arguments

x1, x2	start and end times for the first interval
y1, y2	start and end times for the second interval

Details

Paulo and Oliveira (2004) provide an "overlap rate" statistic for computing the amount of overlap between two (time) intervals. To my knowledge, nobody has described the Overlap Rate in this way, but it is the **Jaccard index** applied to time intervals.

Let $X = [x_{\min}, x_{\max}]$ and $Y = [y_{\min}, y_{\max}]$ be the sets of times spanned by the intervals x and y . Then, $X \cap Y$ is the *intersection* or the times covered by both intervals, and $X \cup Y$ is the *union* or the times covered by either interval. The size of a set A is denoted $|A|$. Then the overlap rate is the Jaccard index or the proportion of elements that the two sets have in common:

$$\text{overlap rate} = \frac{|X \cap Y|}{|X \cup Y|}$$

Value

the overlap rate

References

Paulo, S., & Oliveira, L. C. (2004). Automatic Phonetic Alignment and Its Confidence Measures. In J. L. Vicedo, P. Martínez-Barco, R. Muñoz, & M. Saiz Noeda (Eds.), *Advances in Natural Language Processing* (pp. 36–44). Springer. https://doi.org/10.1007/978-3-540-30228-5_4

Examples

```
compute_overlap_rate(
  c(0.0, 0.0, 0.0, 0.0),
  c(1.0, 1.0, 1.0, NA),
  c(0.5, 2.0, 1.0, 1.0),
  c(2.0, 3.0, 2.0, 2.0)
)
```

```
compute_predictive_value_from_rates
```

Compute positive and negative predictive value

Description

Compute positive and negative predictive value

Usage

```
compute_predictive_value_from_rates(sensitivity, specificity, prevalence)
```

Arguments

sensitivity, specificity, prevalence
vectors of confusion matrix rates

Details

These vectors passed into this function should be some common length and/or length 1. For example, 4 sensitivities, 4 specificities and 1 incidence will work because the sensitivities and specificities have a common length and we can safely recycle (reuse) the incidence value. But 4 sensitivities, 2 specificities, and 1 incidence will *fail* because there is not a common length.

Value

a tibble with the columns **sensitivity**, **specificity**, **prevalence**, **ppv**, **npv** where **ppv** and **npv** are the positive predictive value and the negative predictive value.

Examples

```
compute_predictive_value_from_rates(
  sensitivity = .9,
  specificity = .8,
  prevalence = .05
)
```

```
compute_predictive_value_from_rates(
  sensitivity = .67,
  specificity = .53,
```

```
prevalence = c(.15, .3)
)
```

```
compute_sens_spec_from_ecdf
```

Compute sensitivity and specificity scores from (weighted) observed data

Description

`pROC::roc()` does not support observation-weights when computing ROC curves. This function fills that gap.

Usage

```
compute_sens_spec_from_ecdf(
  data,
  response,
  predictor,
  weights = NULL,
  direction = NULL,
  levels = NULL
)
```

Arguments

data	a dataframe containing responses (groupings) and predictor variable
response	a bare column name with the group status (control vs. cases). If the response has more than two groups, the first element of levels is the control group and the second element of levels is the case group.
predictor	a bare column name with the predictor to use for classification
weights	a bare column name for the observation weights. If weights is set to <code>NULL</code> or not provided, all values receive equal weight and conventional sensitivity and specificity scores are returned.
direction	direction for computing case status. <code>pROC::roc()</code> 's direction conventions are supported: " <code><</code> " (i.e., <code>control < case</code>) and " <code>></code> " (<code>control > case</code>), or " <code>auto</code> " to have <code>pROC</code> guess the direction (if applicable). Alternatively, more verbose directions are supported: " <code>case-low</code> " or " <code>control-high</code> " if a low score predicts case status (<code>case <= threshold < control</code> , analogous to <code>pROC</code> 's " <code>></code> ") and " <code>case-high</code> " or " <code>control-low</code> " if a high score predicts case status (<code>control < threshold <= case</code> , analogous to <code>pROC</code> 's " <code><</code> "). These directions are translated into the <code>pROC</code> conventions.
levels	two-element vector <code>c(control, case)</code> where control is the value of response for the control group and case is the value of response for the case group. The ordering matters: The first element of the vector names the control group.

Details

The `.sensitivities` and `.specificities` columns are calculated directly from the (weighted) ECDFs of the controls and cases, so no call to `pROC::roc()` is made.

`.auc` is calculated using `trapezoid_auc()`.

`c(-Inf, Inf)` are added to the `predictor` vector so that sensitivities and specificities range from 0 to 1.

Value

A dataframe of stepwise empirical ROC coordinates computed from (optionally weighted) data. The output includes columns for the predictor variable, `.sensitivities`, `.specificities`, `.auc`, `.comparison`, `.n_controls`, `.n_cases`, `.w_controls`, `.w_cases`, `.comparison`, `.direction`, `.response`, `.controls`, and `.cases`. `n_` columns contain the number of observations for that predictor value and `w_` contain the total weight of the observations for that predictor value.

Examples

```
# Simulate 3-class dataset
set.seed(100)
n <- 50
means <- c("A" = 0, "B" = 1, "C" = 2)
y <- sample(c("A", "B", "C"), n, replace = TRUE)
x <- rnorm(n, mean = means[y], sd = 1)
w <- runif(n, 0.5, 2)
df <- data.frame(y, x, w)

# Compare "A" (controls) to "C" (cases)
roc_tbl <- compute_sens_spec_from_ecdf(
  data = df,
  response = y,
  predictor = x,
  weights = w,
  direction = "control-low",
  levels = c("A", "C")
)
dplyr::glimpse(roc_tbl)

# Compare "B" (controls) to "C" (cases)
roc_tbl2 <- compute_sens_spec_from_ecdf(
  data = df,
  response = y,
  predictor = x,
  weights = w,
  direction = "control-low",
  levels = c("B", "C")
)
dplyr::glimpse(roc_tbl2)

library(ggplot2)
# Plotting can be tricky bc sens and spec values repeat. geom_path() does the
```

```
# right thing and walks along threshold values to plot the sens-spec pairs
ggplot(roc_tbl) +
  aes(x = .specificities, y = .sensitivities) +
  geom_path() +
  scale_x_reverse() +
  coord_fixed(ratio = 1)
```

```
compute_smooth_density_roc
```

Create an ROC curve from smoothed densities

Description

Create an ROC curve from smoothed densities

Usage

```
compute_smooth_density_roc(
  data,
  controls,
  cases,
  along = NULL,
  best_weights = c(1, 0.5),
  direction = "auto"
)
```

Arguments

<code>data</code>	a dataframe containing densities
<code>controls, cases</code>	bare column names for the densities of the control group and case group
<code>along</code>	optional bare column name for the response values
<code>best_weights</code>	weights for computing the best ROC curve points. Defaults to <code>c(1, .5)</code> , which are the defaults used by <code>pROC::coords()</code> .
<code>direction</code>	direction for computing case status. <code>pROC::roc()</code> 's direction conventions are supported: " <code><</code> " (i.e., <code>control < case</code>) and " <code>></code> " (<code>control > case</code>), or " <code>auto</code> " to have <code>pROC</code> guess the direction (if applicable). Alternatively, more verbose directions are supported: " <code>case-low</code> " or " <code>control-high</code> " if a low score predicts case status (<code>case <= threshold < control</code> , analogous to <code>pROC</code> 's " <code>></code> ") and " <code>case-high</code> " or " <code>control-low</code> " if a high score predicts case status (<code>control < threshold <= case</code> , analogous to <code>pROC</code> 's " <code><</code> "). These directions are translated into the <code>pROC</code> conventions.

Value

the dataframe is updated with new columns for the `.sensitivities`, `.specificities`, `.auc`, `.roc_row`, `.is_best_youden` and `.is_best_closest_topleft`.

Examples

```
set.seed(100)
x1 <- rnorm(100, 4, 1)
x2 <- rnorm(100, 2, .5)
both <- c(x1, x2)
steps <- seq(min(both), max(both), length.out = 200)
d1 <- dnorm(steps, mean(x1), sd(x1))
d2 <- dnorm(steps, mean(x2), sd(x2))
data <- tibble::tibble(
  y = steps,
  d1 = d1,
  d2 = d2,
  outcome = rbinom(200, 1, prob = 1 - (d1 / (d1 + d2))),
  group = ifelse(outcome, "case", "control")
)
compute_smooth_density_roc(data, d1, d2)
compute_smooth_density_roc(data, d1, d2, along = y)

# terrible ROC because the response is not present (just the densities)
data_shuffled <- data[sample(seq_len(nrow(data))), ]
compute_smooth_density_roc(data_shuffled, d1, d2)

# sorted along response first: correct AUC
compute_smooth_density_roc(data_shuffled, d1, d2, along = y)
```

data_acq_consonants	<i>Acquisition and developmental descriptions of consonants and vowels</i>
---------------------	--

Description

This package provides dataframes of information about the consonants and vowels in American English. The following datasets collect *acquisition* (**acq**) features which (try to) characterize the expected acquisition or speech-motor difficulty of speech sounds. See also [data_features_consonants](#).

Usage

```
data_acq_consonants
```

```
data_acq_vowels
```

Format

An object of class **tbl_df** (inherits from **tbl**, **data.frame**) with 24 rows and 16 columns.

An object of class **tbl_df** (inherits from **tbl**, **data.frame**) with 17 rows and 8 columns.

Details

Consonant acquisition features:

`data_acq_consonants` provides the following features:

```
knitr::kable(data_acq_consonants)
```

phone	cmubet	wiscbet	cm2020_90_age_mean	cm2020_90_age_sd	cm2020_90_age_min	cm2020_90_age_max
p	P	p	33.2	6.9	24	36
b	B	b	31.4	7.8	24	36
t	T	t	38.5	9.2	24	36
d	D	d	35.7	6.7	24	36
k	K	k	37.7	7.3	24	36
g	G	g	36.8	6.6	24	36
tʃ	CH	tsh	53.5	10.7	36	60
dʒ	JH	dzh	51.0	11.8	36	60
m	M	m	33.2	6.7	24	36
n	N	n	33.1	7.4	24	36
ŋ	NG	ng	40.3	10.8	24	36
f	F	f	38.3	6.3	24	36
v	V	v	50.8	10.8	36	60
θ	TH	th	77.0	7.4	72	84
ð	DH	dh	69.0	11.3	54	60
s	S	s	51.3	16.3	24	36
z	Z	z	56.8	14.3	30	36
	SH	sh	55.0	10.5	36	60
	ZH	zh	70.7	12.2	60	60
h	HH	h	35.0	7.0	24	36
l	L	l	53.8	10.4	24	36
r	R	r	66.6	18.6	30	36
w	W	w	35.2	6.8	24	36
j	Y	j	45.8	11.0	30	36

Description of each column:

phone phone in IPA

cmubet phone in the CMU alphabet

wiscbet phone in an older system used by our lab

cm2020_90_age_mean, cm2020_90_age_sd, cm2020_90_age_min, cm2020_90_age_max

Age of acquisition statistics reported by Crowe & McLeod (2020). Statistics are the mean, SD, min and max age (in months) when children reached 90% accuracy on a consonant.

cm2020_90_num_studies Number of studies used by Crowe & McLeod (2020) to compute the corresponding statistics.

cm2020_90_stage Developmental stage assigned to the consonant by Crowe & McLeod (2020). Sounds with an **age_mean** before 48 months are **early**, before 60 months are **middle**, and of 60 or older are **late**.

s93_eights Developmental stage of Shriberg (1993)—that is, the **early** 8, **middle** 8 and **late** 8 consonants.

k1992_set Developmental set from Kent (1992). Sets corresponds to the age of 90% mastery in Sander (1972): Set 1 is mastered at age 3-years-old, Set 2 at age 4, Set 3 at age 6, and Set 4 at a later age.

kd2018_complexity Phonetic complexity scores from Kuruvilla-Dugdale et al. (2018). This scoring system is based on the development description of vowels and consonants in Kent (1992). The scores for individual segments range from 1 for the earliest vowels to 6 for the last-acquired consonants. Under this system, assign a score to each part of a syllable (onset, nucleus, coda) using these scores when the syllable part is a single segment and using scores of 7 and 8 for 2-consonant and 3-consonant clusters, respectively.

hml84_frequency, hml84_log10fpm Raw frequency and log10 frequency per million of the phoneme in the Hoosier Mental Lexicon (Nusbaum, Pisoni, Pisoni, 1984) word-frequency dictionary.

mhr82_frequency, mhr82_log10fpm Raw frequency and log10 frequency per million of the phoneme in the Moe, Hopkins, and Rush (1982) word frequency dictionary of first-graders.

Vowel acquisition features:

data_acq_vowels provides the following features:

```
knitr::kable(data_acq_vowels)
```

phone	cmubet	wiscbet	kd2018_complexity	hml84_frequency	hml84_log10fpm	mhr82_frequency	mhr82_log10fpm
i	IY	i	2	82430	4.487818	28537	4.450000
	IH	I	4	195348	4.862542	43999	4.642542
e	EY	eI	4	38966	4.162419	16679	4.216242
	EH	E	3	73107	4.435692	31322	4.493569
æ	AE	ae	4	106838	4.600459	40290	4.600459
	AH	^	1	44555	4.220629	22826	4.352063
ə	AH	4	1	231568	4.936412	38977	4.589364
u	UW	u	2	57320	4.330039	19188	4.280039
	UH	U	4	13557	3.703897	4288	3.630389
o	OW	oU	2	61954	4.363802	17330	4.236380
	AO	c	3	26110	3.988540	12004	3.888540
	AA	@	1	36589	4.135084	17326	4.235084
a	AW	@U	3	16079	3.777992	7377	3.867992
a	AY	@I	3	39275	4.165849	23807	4.375849
	OY	cI	3	2140	2.902147	744	2.852147
	ER	3^	5	17048	3.803406	4227	3.623406
	ER	4^	5	41966	4.194631	10676	4.014631

phone phone in IPA

cmubet phone in the CMU alphabet

wiscbet phone in an older system used by our lab

kd2018_complexity Phonetic complexity scores from Kuruvilla-Dugdale et al. (2018).

This scoring system is based on the development description of vowels and consonants in Kent (1992). The scores for individual segments range from 1 for the earliest vowels to 6 for the last-acquired consonants. Under this system, assign a score to each part of a syllable (onset, nucleus, coda) using these scores when the syllable part is a single segment and using scores of 7 and 8 for 2-consonant and 3-consonant clusters, respectively.

hml84_frequency, hml84_log10fpm Raw frequency and log10 frequency per million of the phoneme in the Hoosier Mental Lexicon (Nusbaum, Pisoni, Pisoni, 1984) word-frequency dictionary.

mhr82_frequency, mhr82_log10fpm Raw frequency and log10 frequency per million of the phoneme in the Moe, Hopkins, and Rush (1982) word frequency dictionary of first-graders.

Crowe and McLeod (2020) norms for English consonant acquisition:

Crowe and McLeod (2020, below as the `cm2020_` variables) provides a systematic review and summary statistics for age of acquisition norms for English consonants. They scoured the literature of acquisition ages for individual consonants and computed summary statistics on them. They considered just accuracy of sounds when produced in single words. Their sources include a mix of a journal articles and norms for articulation assessments. They do not weight statistics from individual studies by sample size or sampling procedure.

I prepared the Crowe and McLeod (2020) data by copying the relevant numbers from their Table 2 making the following changes: 1) rounding mean and SD values to 1 decimal point (3 days for ages in months), 2) dropping / /, 3) using /r/, /g/, /t /, /d / for IPA characters instead of the specialized characters used in the article.

English language phoneme frequencies:

The `hml84_frequency` column provides the frequency count for the phonemes in the Hoosier Mental Lexicon (Nusbaum, Pisoni, Pisoni, 1984). That is, we count how many times the phonemes appear in each word in the word list and weight them by the word frequency. For example, "ad" has two phonemes and a corpus frequency of 99, so it counts for 99 /æ/ tokens and 99 /d/ tokens.

The HML frequency counts derive from the Brown Corpus of one million English words that were printed/published in 1961. The HML provides frequencies of *phonological* words, and homophones are combined into a single entry. For example, the word "ad" has a frequency of 99 (11 *ad* tokens plus 88 *add* tokens). That's why, I suppose, it's a *mental* lexicon. Approximately 8,000 words in the HML were not in the K&F frequency word list, and these are apparently assigned a frequency of 1.

The `mhr82_frequency` column was constructed in a similar way but the frequencies were based on a corpus of words used by first-graders (Moe, Hopkins, & Rush, 1982).

The `hml84_log10fpm` and `mhr82_log10fpm` columns provide the frequency in log-10 frequency per million which is more appropriate for analyses. Computing frequency per million normalize the frequency counts across different corpora, and log-frequency is better suited than raw or normalized frequency counts.

I computed these phoneme frequencies independently, but retrieved my copies of the HML and MHR frequency-pronunciation tables from [a course](#) by Smith, Beckman and Foltz (2016).

The early 8, middle 8 and late 8 (Shriberg, 1993):

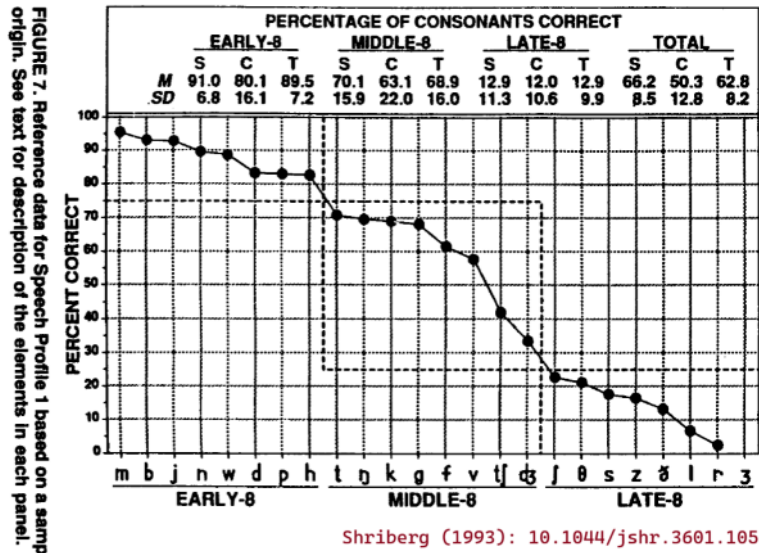
The English consonants are often broken down into three developmental classes, based on Shriberg (1993):

- Early 8: m b j n w d p h
- Middle 8: t ɲ k g f v t d ,
- Late 8: s z ð l r

This classification is included as the `s93_eights` column.

From these names alone, we might interpret these classes such that sounds in the Early 8 would be acquired before the ones in the Middle 8, and likewise that the Middle 8 would be acquired before the Late 8. But these classes were not created by examining patterns of typical consonant acquisition.

For some context, Shriberg (1993) introduces the Early 8, Middle 8, and Late 8 data by describing the following panel of the article's Figure 7:



About which, Shriberg (1993) says: "The values for this trend, which is a profile of consonant mastery, were taken from a group of 64 3- to 6-year-old speech-delayed children (Shriberg, Kwiatkowski, & Gruber, 1992). Severity of involvement of the 24 English consonants is represented as the percentage correct for each consonant sorted in decreasing order from left to right. Notice that the most obvious breaks in this function allow for a division of the 24 consonants into three groups of eight sounds termed the **Early-8**, averaging over 75% correct, the **Middle-8**, averaging 25%-75% correct, and the **Late-8**, including consonants averaging less than 25% correct in continuous conversational speech (/ / is infrequently represented in young, speech-delayed children's spontaneous conversational speech)."

So, there were 64 3–6-year-old children with speech delays, and consonant sounds were divided into three classes based on how often *these children* produced the sounds correctly on average in a conversational speech sample. This classification is not so much a measure of the relative ordering of speech sound development as it is **the relative difficulty of these sounds for children with a speech delay of unknown origin**. It would be more appropriate to replace the levels of Early/Middle/Late with Easy/Medium/Hard.

Phonetic complexity (Kent, 1992; Kuruvilla-Dugdale et al. 2018):

Phonetic complexity measures (`k1992_set` and `kd2018_complexity`) assign the speech sounds different complexity levels based on biological principles outlined in Kent (1992). Because Kent (1992) is a book chapter that is not floating around online, it's worthwhile to review the provenance of these complexity measures. In short, Kent (1992) applied interpreted consonant and vowel development data in terms of their motor demands.

Sander (1972) set out to construct a set of developmental norms for typical consonant acquisition in English. His big idea was to include the median age of acquisition as well as the 90th percentile age of acquisition. The median can tell us something about the average acquisition of the speech sounds, and the 90th percentile can set a benchmark for delayed acquisition. There are some quirks of the methodology. First, Sander (1972) was targeting "customary articulation" which was defined using production accuracy average across word positions. So, the age of 50% customary articulation for /t/ is the earliest age when the average of word-initial accuracy, word-medial accuracy and word-final accuracy is greater than 50%. Second, the norms for this study were created by augmenting data from 3–8-year-olds (Templin, 1957; $n = 480$) with some earlier data for 2-year-olds (Wellman et al. 1931; $n = 15$).

Sander (1972) presented these acquisition norms in the following figure:

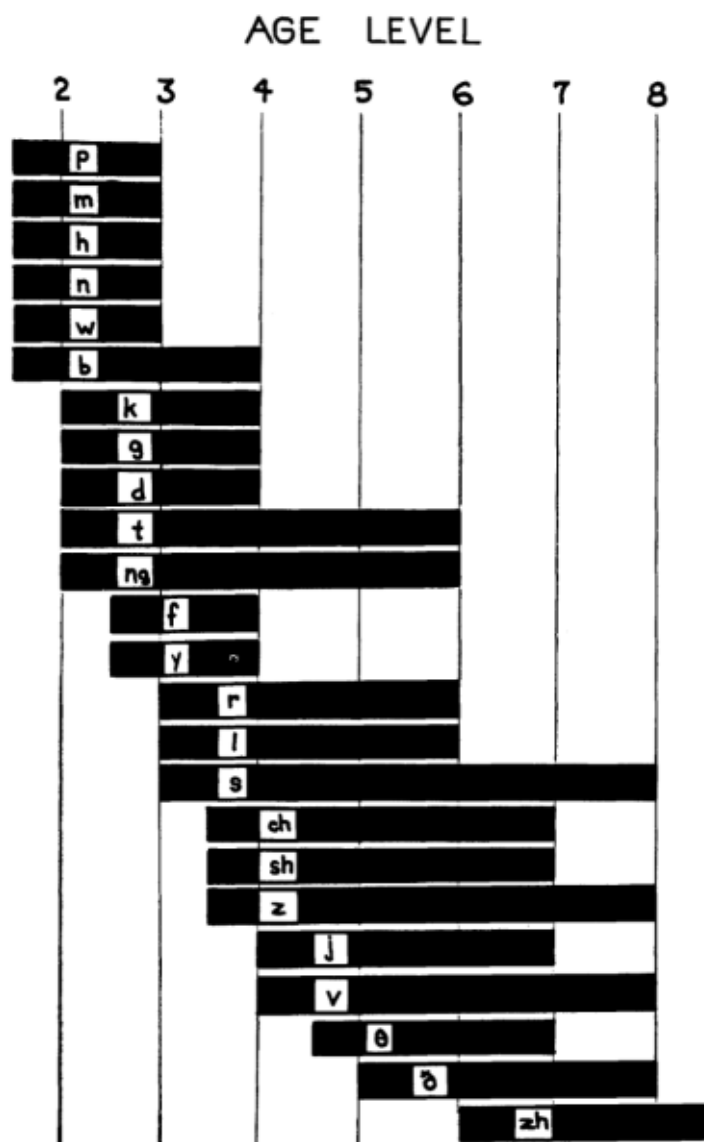


Figure 1. Average age estimates and upper age limits of customary consonant production. The solid bar corresponding to each sound starts at the median age of customary articulation; it stops at an age level at which 90% of all children are customarily producing the sound. (From Templin, 1957; Wellman et al., 1931.)

Sander (1972): 10.1044/jshd.3701.55

Kent (1992) aimed to explain the course of English sound development in terms of biological and motoric principles. He examined the ages of 90% acquisition from Sander (1972)—that is, the right edges of the bars in the previous figure—and observed that /p m n w h/ are mastered at age 3, /b d k g j f/ at age 4, /t ŋ r l/ at age 6 and /s z v θ ð t d/ after age 7. He then described motoric demands in each of these sets of sounds. I'll paraphrase:

- Set 1 requires fast "ballistic" movements for stops /p m n/, slow "ramp" movements for /w h/, velopharyngeal control for oral-nasal contrast, laryngeal control for voicing contrast.
- Set 2 adds more stops /b d k g/ and another ramp /j/ and a new place of articulation (velars), but also requires "fine force regulation for frication" for /f/.
- Set 3 adds more stops /t ɲ/, but also requires tongue "bending" for /r/ and /l/.
- Set 4 adds more lingual fricatives /s z ð/ which require tongue bending and fine force control along with /v t d/. Kent does not characterize the motor demands for the affricates /t d/.

Let's pause for a moment and observe that this breakdown is just an attempt to describe the Sander (1972) norms, and it is somewhat underdeveloped. For example, why is /t/ in Set 3 but /d/ in Set 2? It is not answered here, but I think this late mastery is an artefact of Sander's requirement of 90% accuracy averaging over the three word-positions. The medial and final productions of /t/ might require allophonic variation in /t/ (e.g., flapping or glottalization), so mastery of /t/ would require different motor gestures and some phonological knowledge on the part of the child. But in Kent's description /t/ is a later-mastered ballistic movement.

Still, the main point of Kent's description, I think, is that lingual (tongue) consonants are more difficult. Elsewhere in the chapter, Kent (1992) describes how the tongue is a "muscular hydrostat" like an elephant trunk, and bending a hydrostat requires coordination of different muscle directions:

"Gaining motor control over a hydrostat presents some special problems to the young child learning speech. For one, bending the hydrostat is unlike bending a jointed structure such as a finger. The tongue has no joints per se; it flexes by appropriate contraction of its three-dimensional network of intrinsic longitudinal, vertical, and transverse fibers. Bending a hydrostat requires that muscle fibers be shortened on one aspect simultaneously with a resistance to a change in diameter (Smith and Kier 1989). If the diameter change is not resisted, then the hydrostat will shorten on one side but will not bend. To use the tongue in speech, the child must learn to control the tongue to meet skeletal, movement, and shaping requirements, often simultaneously. These special characteristics of the tongue may well play a role in vowel and consonant mastery."

Kim and colleagues (2010) applied these developmental sets (`k1992_set`) as articulatory complexity *levels* while examining consonant errors in dysarthric speech. They then asked questions such as whether more complex consonants had more consonant errors than less complex ones (*yes*) or whether lower intelligibility speakers made more complexity-reducing consonant substitutions than higher intelligibility speakers (*apparently so*). Examining the speech of 5-year-olds, Allison and Hustad (2014) later used these complexity levels as a way to *score* the phonetic complexity of sentences. They assigned consonants 1–5 scores (the 1–4 complexity levels with a score of 5 for consonants clusters), and summed up the scores to provide a complexity score for a sentence. Three of the eight 5-year-olds with dysarthria showed a negative effect of sentence complexity on intelligibility.

Kent (1992) also described the yearly developmental progression of vowels. I'll paraphrase again:

- By age 1: Infants produce *vocants* (vowel precursors) which correspond to the low-front, central and low-back vowels /æ ə/. Thus, the tongue only moves in the anterior-posterior direction (i.e., there is limited up-down movement).

- By age 2: Toddlers produce the “maximally dissimilar” corner vowels /i u / and produce /o/ and the central vowels / ə/.
- By age 3: Children incorporate two lower vowels / / and the diphthongs /a a / which require gliding movements.
- By age 4: Children incorporate the remaining non-rhotic vowels / e æ/. The appearance of the front vowels suggests that tongue-jaw coordination is a relatively late motor achievement. (/i/ appears earlier because its extreme height is easy.)
- Lastly: Children incorporate / / last because these r-colored vowels require tongue bending.

Kuruvilla-Dugdale and colleagues (2018) used this description to incorporate vowels into the phonetic complexity scale (kd2018_complexity). The / ə / vocants from age 1 and the vowels from age 2 mark the bottom of the complexity scale. The vowels that are acquired at ages 3, 4 and afterwards are assigned to the consonant complexity levels with the same age of mastery. Finally, consonant clusters serve as the ceiling for the scale:

Table: Phonetic complexity scores from Kuruvilla-Dugdale et al. (2018).

kd2018_complexity	consonants	vowels
1		ə
2		i u o
3	p m n h w	a a
4	b d k g f j	e æ
5	t ŋ l r	
6	t d v ð s z	
7	2-consonant clusters	
8	3-consonant clusters	

It is not clear how to apply this scale, so my approach has been to break words into subsyllabic units and assign scores to the syllable onsets, nucleui and codas in each word. For example, "jump" is /d / + / / + /mp/ so it would have complexity of 6 + 1 + 7 = 14, and "jumper" includes a syllable break between the cluster, so it would have a score 6 + 1 + 1 + 1 + 5 = 14.

Kuruvilla-Dugdale and colleagues (2018) used this scoring system to compare intelligibility for low complexity versus high complexity words. For example, for speakers with ALS and mild dysarthria, there was statistically clear reduction in intelligibility for high complexity words but not for low complexity words. I applied this scoring system on single-word intelligibility in children’s speech (Mahr & Hustad, 2023). There was a probable but not statistically clear negative effect of complexity on intelligibility over and above the effects of age, word frequency and word neighborhood competition. (Regrettably, I coded the one consonant cluster in the word list with a complexity of 8 instead of 7, but otherwise this is the approach.)

References

Allison, K. M., & Hustad, K. C. (2014). Impact of sentence length and phonetic complexity on intelligibility of 5-year-old children with cerebral palsy. *International Journal of Speech-Language Pathology*, 16(4), 396–407. <https://doi.org/10.3109/17549507.2013.876667>

- Crowe, K., & McLeod, S. (2020). Children’s English Consonant Acquisition in the United States: A Review. *American Journal of Speech-Language Pathology*, 29(4), 2155–2169. https://doi.org/10.1044/2020_AJSLP-19-00168
- Kent, R. D. (1992). The Biology of Phonological Development. In C. A. Ferguson, L. Menn, & C. Stoel-Gammon (Eds.), *Phonological development: Models, research, implications* (pp. 65–90). York Press.
- Kim, H., Martin, K., Hasegawa-Johnson, M., & Perlman, A. (2010). Frequency of consonant articulation errors in dysarthric speech. *Clinical Linguistics and Phonetics*, 24(10), 759–770. <https://doi.org/10.3109/02699206.2010.497238>
- Kuruvilla-Dugdale, M., Custer, C., Heidrick, L., Barohn, R., & Govindarajan, R. (2018). A Phonetic Complexity-Based Approach for Intelligibility and Articulatory Precision Testing: A Preliminary Study on Talkers With Amyotrophic Lateral Sclerosis. *Journal of Speech, Language, and Hearing Research*, 61(9), 2205–2214. https://doi.org/10.1044/2018_JSLHR-S-17-0462
- Mahr, T. J., & Hustad, K. C. (2023). Lexical Predictors of Intelligibility in Young Children’s Speech. *Journal of Speech, Language, and Hearing Research*, 66(8S), 3013–3025. https://doi.org/10.1044/2022_JSLHR-22-00294
- Sander, E. K. (1972). When are Speech Sounds Learned? *Journal of Speech and Hearing Disorders*, 37(1), 55–63. <https://doi.org/10.1044/jshd.3701.55>
- Shriberg, L. D. (1993). Four New Speech and Prosody-Voice Measures for Genetics Research and Other Studies in Developmental Phonological Disorders. *Journal of Speech, Language, and Hearing Research*, 36(1), 105–140. <https://doi.org/10.1044/jshr.3601.105>

data_example_intelligibility_by_length

Simulated intelligibility scores by utterance length

Description

A dataset of simulated intelligibility scores for testing and demonstrating modeling functions. These were created by fitting a Bayesian model of the raw Hustad and colleagues (2020) and drawing 1 sample from the posterior distribution of expected predictions (i.e., “epreds”). In other words, these values are model predictions of the original dataset. They are correlated with original dataset values at $r = .86$. We might think of the simulation as adding random noise to the original dataset.

Usage

data_example_intelligibility_by_length

Format

A data frame with 694 rows and 5 variables:

child identifier for the child

age_months child’s age in months

length_longest length of the child's longest utterance

tocs_level utterance length

sim_intelligibility child's intelligibility for the given utterance length (proportion of words said by the child that were correctly transcribed by two listeners)

References

Hustad, K. C., Mahr, T., Natzke, P. E. M., & Rathouz, P. J. (2020). Development of Speech Intelligibility Between 30 and 47 Months in Typically Developing Children: A Cross-Sectional Study of Growth. *Journal of Speech, Language, and Hearing Research*, 63(6), 1675–1687. https://doi.org/10.1044/2020_JSLHR-20-00008

data_fake_intelligibility

Fake intelligibility data

Description

A dataset of fake intelligibility scores for testing and demonstrating modeling functions. These were created by randomly sampling 200 rows of an intelligibility dataset and adding random noise to the **age_months** and **intelligibility** variables. These values do not measure any real children but represent plausible age and intelligibility measurements from our kind of work.

Usage

data_fake_intelligibility

Format

A data frame with 200 rows and 2 variables:

age_months child's age in months

intelligibility child's intelligibility (proportion of words said by the child that were correctly transcribed by two listeners)

data_fake_rates	<i>Fake speaking rate data</i>
-----------------	--------------------------------

Description

A dataset of fake speaking rate measures for testing and demonstrating modeling functions. These were created by randomly sampling 200 rows of a speaking rate dataset and adding random noise to the **age_months** and **speaking_sps** variables. These values do not measure any real children but represent plausible age and rate measurements from our kind of work.

Usage

```
data_fake_rates
```

Format

A data frame with 200 rows and 2 variables:

age_months child's age in months

speaking_sps child's speaking rate in syllables per second

data_features_consonants	<i>Phonetic features of consonants and vowels</i>
--------------------------	---

Description

This package provides dataframes of information about the consonants and vowels in American English. The *phonetic* features are conventional descriptions of how sounds are produced. See also [data_acq_consonants](#).

Usage

```
data_features_consonants
```

```
data_features_vowels
```

Format

An object of class **tbl_df** (inherits from **tbl**, **data.frame**) with 24 rows and 11 columns.

An object of class **tbl_df** (inherits from **tbl**, **data.frame**) with 17 rows and 13 columns.

Details

Most of the phonetic features are self-evident and definitional. For example, /p/ is *the* bilabial voiceless stop. For fuzzier features, I consulted the general IPA chart and the Wikipedia page on English phonology. These issues included things like: *what are the lax vowels again?* or *the last two rows of the consonant tables are approximants, so /r,l,j/ are approximants.*

Some features have alternative feature sets in order to accommodate degrees of aggregation. For example, /r,l,j,w/ are *approximant* in **manner** but divided into *liquid* and *glide* in **manner_alt**.

Phonetic features of consonants:

data_features_consonants is a dataframe with 24 rows and 10 variables.

```
knitr::kable(data_features_consonants)
```

phone	cmubet	wiscbet	voicing	voicing_alt	manner	manner_alt	place	place_fct
p	P	p	voiceless	spread_glottis	stop	stop	labial	labial
b	B	b	voiced	plain	stop	stop	labial	labial
t	T	t	voiceless	spread_glottis	stop	stop	alveolar	alveolar
d	D	d	voiced	plain	stop	stop	alveolar	alveolar
k	K	k	voiceless	spread_glottis	stop	stop	velar	velar
g	G	g	voiced	plain	stop	stop	velar	velar
t	CH	tsh	voiceless	spread_glottis	affricate	affricate	postalveolar	postalveolar
d	JH	dzh	voiced	plain	affricate	affricate	postalveolar	postalveolar
m	M	m	voiced	NA	nasal	nasal	labial	labial
n	N	n	voiced	NA	nasal	nasal	alveolar	alveolar
ŋ	NG	ng	voiced	NA	nasal	nasal	velar	velar
f	F	f	voiceless	spread_glottis	fricative	fricative	labiodental	labiodental
v	V	v	voiced	plain	fricative	fricative	labiodental	labiodental
	TH	th	voiceless	spread_glottis	fricative	fricative	dental	dental
ð	DH	dh	voiced	plain	fricative	fricative	dental	dental
s	S	s	voiceless	spread_glottis	fricative	fricative	alveolar	alveolar
z	Z	z	voiced	plain	fricative	fricative	alveolar	alveolar
	SH	sh	voiceless	spread_glottis	fricative	fricative	postalveolar	postalveolar
	ZH	zh	voiced	plain	fricative	fricative	postalveolar	postalveolar
h	HH	h	voiceless	spread_glottis	fricative	fricative	glottal	glottal
l	L	l	voiced	NA	approximant	liquid	alveolar	alveolar
r	R	r	voiced	NA	approximant	liquid	postalveolar	postalveolar
w	W	w	voiced	NA	approximant	glide	labiovelar	NA
j	Y	j	voiced	NA	approximant	glide	palatal	palatal

Description of each column:

phone phone in IPA

cmubet phone in the CMU alphabet

wiscbet phone in an older system used by our lab

voicing *voiced* versus *voiceless*

voicing_alt *spread_glottis* versus *plain*

manner manner of articulation

manner_alt alternative manner coding that separates *approximants* into *liquids* and *glides*

place place of articulation

place_fct place coded as a factor and ordered based on frontness of the articulators. *labiovelar* is recoded as NA.

sonorance *obstruent* versus *sonorant*

sonorance_alt *obstruant* versus *sonorant* versus *strident*.

Levels of the factor columns:

```
data_features_consonants |>
  lapply(levels) |>
  Filter(length, x = _)
#> $place_fct
#> [1] "labial"          "labiodental"    "dental"         "alveolar"       "postalveolar"
#> [6] "palatal"         "velar"          "glottal"
```

Considerations about consonant phonetic features:

The CMU alphabet does not include a glottal stop.

Here /f,v/ are coded as *strident* following [Wikipedia](#) and *Sound Pattern of English*. If this feature value doesn't seem right, we should probably use an alternative feature of *sibilant* for the stridents minus /f,v/.

The alternative voicing scheme was suggested by a colleague because of how the voice-voiceless phonetic contrast is achieved with different articulatory strategies in different languages. Note that **voicing_alt** does not assign a feature to nasals or approximants.

Phonetic features of vowels:

data_features_vowels is a dataframe with 17 rows and 11 variables.

```
knitr::kable(data_features_vowels)
```

phone	cmubet	wiscbet	hint	manner	manner_alt	tenseness	height	height_fct	height_alt
i	IY	i	beat	vowel	vowel	tense	high	high	high
	IH	I	bit	vowel	vowel	lax	mid-high	mid-high	high
e	EY	eI	bait	vowel	vowel	tense	mid-high	mid-high	mid
	EH	E	bet	vowel	vowel	lax	mid-low	mid-low	mid
æ	AE	ae	bat	vowel	vowel	lax	low	low	low
	AH	^	but	vowel	vowel	lax	mid-low	mid-low	mid
ə	AH	4	comma	vowel	vowel	lax	mid-low	mid-low	mid
u	UW	u	boot	vowel	vowel	tense	high	high	high
	UH	U	book	vowel	vowel	lax	mid-high	mid-high	high
o	OW	oU	boat	vowel	vowel	tense	mid-high	mid-high	mid
	AO	c	bought	vowel	vowel	tense	mid-low	mid-low	low
	AA	@	bot	vowel	vowel	tense	low	low	low
a	AW	@U	bout	vowel	diphthong	diphthong	diphthong	NA	diphthong
a	AY	@I	bite	vowel	diphthong	diphthong	diphthong	NA	diphthong

OY	cI	boy	vowel	diphthong	diphthong	diphthong	NA	diphthong
ER	3 [^]	letter	vowel	r-colored	r-colored	mid-low	mid-low	mid
ER	4 [^]	burt	vowel	r-colored	r-colored	mid-low	mid-low	mid

Description of each column:

phone phone in IPA

cmubet phone in the CMU alphabet

wiscbet phone in an older system used by our lab

hint a word containing the selected vowel

manner manner of articulation

manner_alt alternative manner with *vowel*, *diphthong* and *r-colored*

tenseness *tense* versus *lax* (versus *diphthong* and *r-colored*)

height vowel height on a four-level scale

height_fct height coded as a factor ordered *high*, *mid-high*, *mid-low*, *low*. *diphthong* is recoded to NA.

height_alt vowel height on a three-level scale

backness vowel backness

backness_fct backness coded as a factor ordered *front*, *central*, *back*. *diphthong* is recoded to NA.

rounding *unrounded* versus *rounded* (versus *diphthong* and *r-colored*)

Levels of the factor columns:

```
data_features_vowels |>
  lapply(levels) |>
  Filter(length, x = _)
#> $height_fct
#> [1] "high"      "mid-high" "mid-low"  "low"
#>
#> $backness_fct
#> [1] "front"    "central"  "back"
```

Considerations about vowel features:

I don't consider /e / and /o / to be diphthongs, but perhaps **manner_alt** could encode the difference of these vowels from the others.

In the CMU alphabet and ARPAbet, vowels can include a number to indicate vowel stress, so AH1 or AH2 is / / but AH0 is /ə/.

The vowel features for General American English, [according to Wikipedia](#), are as follows:

I adapted these features in this way:

- A four-level breakdown of height (high, mid-high, mid-low, low) was used instead of a three-level one (close, mid, open).
- *tense* and *lax* features were directly borrowed. Diphthongs and r-colored vowels were not assign a tenseness.
- / / moved to *back* (following the general IPA)
- diphthongs have no backness or height

- r-colored vowels were given the backness and height of the /,ə/

Based on the assumption that /,ə/ are the same general vowel with differing stress, these vowels have the same features. This definition clashes with the general IPA chart which places / / as a back vowel. However, / / is a conventional notation. Quoting [Wikipedia](#) again: "Although the notation / / is used for the vowel of STRUT in RP and General American, the actual pronunciation is closer to a near-open central vowel [] in RP and advanced back [] in General American." That is, / / is fronted in American English (hence, *mid*) in American English.

data_tocs_items

Items from the TOCS+ used by the WISC Lab

Description

We use items from the Test of Children's Speech (TOCS+, Hodge & Gotzke, 2014) in our child speech and listener intelligibility studies.

Usage

data_tocs_items

Format

A data frame with 103 rows and 4 variables:

tocs_type whether the item is a **single-word** or **multiword** utterance

tocs_level number of words in the prompt

tocs_item identifier for the item (e.g., within filenames)

tocs_prompt the actual prompt presented to the child

tocs_item_note any notes about an item's usage in experiments or analyses

These are the items:

tocs_type	tocs_level	tocs_item	tocs_prompt
multiword	2	S2T01	Bird house
multiword	2	S2T02	Cowboy boots
multiword	2	S2T03	Hug daddy
multiword	2	S2T04	Get off
multiword	2	S2T05	Animal crackers
multiword	2	S2T06	Living room
multiword	2	S2T07	Doctor bag
multiword	2	S2T08	Baby sock
multiword	2	S2T09	Big potato
multiword	2	S2T10	Say goodbye
multiword	3	S3T01	Gather the toys
multiword	3	S3T02	Five more cookies

multiword	3	S3T03	Make a birdhouse
multiword	3	S3T04	Tie those shoes
multiword	3	S3T05	Put these together
multiword	3	S3T06	That's not white
multiword	3	S3T07	Open the house
multiword	3	S3T08	Take his turn
multiword	3	S3T09	Pull his tooth
multiword	3	S3T10	Point to Teddy
multiword	3	S3T11	Cook big hotdogs
multiword	4	S4T01	She could hide here
multiword	4	S4T02	Wear a cowboy hat
multiword	4	S4T03	Find the brown lid
multiword	4	S4T04	Both faces are happy
multiword	4	S4T05	That is her dog
multiword	4	S4T06	He likes potato chips
multiword	4	S4T07	Jump over the box
multiword	4	S4T08	She chewed her fingers
multiword	4	S4T09	Cut two small pieces
multiword	4	S4T10	Get them some coffee
multiword	5	S5T01	Baby likes his new toy
multiword	5	S5T02	They'll eat those hotdogs soon
multiword	5	S5T03	His fingers are in wrong
multiword	5	S5T04	They are singing Happy Birthday
multiword	5	S5T05	Water shoots from that gun
multiword	5	S5T06	This cheese doesn't smell good
multiword	5	S5T07	The sign says keep out
multiword	5	S5T08	Tie up the garbage bag
multiword	5	S5T09	Give the flowers some water
multiword	5	S5T10	Put all the toys away
multiword	6	S6T01	He wants somebody to push him
multiword	6	S6T02	He gave Mommy a birthday card
multiword	6	S6T03	That baby is learning to talk
multiword	6	S6T04	She laughed at his funny jokes
multiword	6	S6T05	There's a playground near the school
multiword	6	S6T06	He winds up the toy ghost
multiword	6	S6T07	Make a check beside the two
multiword	6	S6T08	The cowboy hat has a feather
multiword	6	S6T09	Taste the cookies that she baked
multiword	6	S6T10	Take the lid off the pot
multiword	7	S7T01	Pour some tea for both of us
multiword	7	S7T02	The loud noise scared the cat away
multiword	7	S7T03	Don't let go or it will fall
multiword	7	S7T04	She is showing him how to jump
multiword	7	S7T05	Baby reached to get another chocolate cookie
multiword	7	S7T06	The bird landed beside the shaggy dog
multiword	7	S7T07	She got mad and pushed the boy
multiword	7	S7T08	They ate birthday cake and drank pop
multiword	7	S7T09	Be very quiet when Baby is sleeping

multiword	7	S7T10	Fill the fridge with things to eat
single-word	1	WT01	Pizza
single-word	1	WT02	Hug
single-word	1	WT03	Boot
single-word	1	WT04	Ball
single-word	1	WT05	Bee
single-word	1	WT06	Buddy
single-word	1	WT07	Hoot
single-word	1	WT08	Mud
single-word	1	WT09	A dress
single-word	1	WT10	Snow
single-word	1	WT11	Chew
single-word	1	WT12	Pan
single-word	1	WT13	Come
single-word	1	WT14	Boy
single-word	1	WT15	Bad
single-word	1	WT16	Hot
single-word	1	WT17	Gum
single-word	1	WT18	Dee
single-word	1	WT19	Beanie
single-word	1	WT20	Zoo
single-word	1	WT21	Hat
single-word	1	WT22	Seat
single-word	1	WT23	Yawn
single-word	1	WT24	No
single-word	1	WT25	Full
single-word	1	WT26	Bus
single-word	1	WT27	Jar
single-word	1	WT28	Top
single-word	1	WT29	Lock
single-word	1	WT30	Walk
single-word	1	WT31	Bow
single-word	1	WT32	Eat
single-word	1	WT33	Sheet
single-word	1	WT34	Rock
single-word	1	WT35	House
single-word	1	WT36	Pipe
single-word	1	WT37	Toys
single-word	1	WT38	Type
single-word	1	WT39	Wait
single-word	1	WT40	Whip
single-word	1	WT41	mIm
single-word	1	WT42	Big

These are the items with entries in the "tocs_item_note" fields:

tocs_type	tocs_level	tocs_item	tocs_prompt	tocs_item_note
-----------	------------	-----------	-------------	----------------

multiword	3	S3T09	Pull his tooth	possibly not an original TOCS item
multiword	3	S3T11	Cook big hotdogs	practice item for intelligibility studies (usually not
multiword	5	S5T10	Put all the toys away	practice item for intelligibility studies (usually not
single-word	1	WT01	Pizza	practice item for intelligibility studies (usually not
single-word	1	WT02	Hug	practice item for intelligibility studies (usually not
single-word	1	WT09	A dress	exclude from intelligibility analyses for being two w
single-word	1	WT18	Dee	sometimes exclude from intelligibility analyses for n
single-word	1	WT31	Bow	rhymes with 'snow'
single-word	1	WT35	House	noun; possibly not an original TOCS item
single-word	1	WT36	Pipe	possibly not an original TOCS item
single-word	1	WT37	Toys	possibly not an original TOCS item
single-word	1	WT38	Type	possibly not an original TOCS item
single-word	1	WT39	Wait	possibly not an original TOCS item
single-word	1	WT40	Whip	possibly not an original TOCS item
single-word	1	WT41	mIm	exclude from intelligibility analyses for being a non
single-word	1	WT42	Big	possibly not an original TOCS item

Details

Most of these items are subset from a larger pool of items on the TOCS+. Hustad and colleagues (2021) is a supplemental material for a larger study about speech intelligibility which describes where the items were taken from within the larger TOCS+ task. (It is unclear, as I write this in 2026, which items were added by our group to the TOCS+ pool, so I have added "possibly not an original TOCS item" to items that do not appear in that supplemental material.)

In this picture-prompted repetition task, a child hears a *prompt*, repeats it (*production*), and later on listeners transcribe it (*transcription*). In our listener-response database, we differentiate these three forms with fields named **tocs_prompt** (prompt), **sentence** (production), and **response** (transcription).

References

Hodge, M. M., & Gotzke, C. (2014). Construct-related validity of the TOCS+ measures: Comparison of intelligibility and speaking rate scores in children with and without speech disorders. *Journal of Communication Disorders*, 51, 51–63. <https://doi.org/10.1016/j.jcomdis.2014.06.007>

Hustad, K. C., Mahr, T. J., Natzke, P., & Rathouz, P. J. (2021). Supplemental Material 1: Intelligibility growth between 30 and 119 months (Hustad et al., 2021) (Version 1). ASHA Journals. <https://doi.org/10.23641/asha.16583426.v1>

Description

`dir_sync_down()` syncs the files in one directory into another directory. Files that are present in `path` but missing from `path_new` are copied. Files that are present in both directories but differ are overwritten. Files that are present in `path_new` but absent from `path` are ignored unless `.delete = TRUE`.

Usage

```
dir_sync_down(
  path,
  path_new,
  .dry_run = TRUE,
  .delete = FALSE,
  .compare = c("metadata", "md5", "xxhash")
)
```

Arguments

<code>path</code>	Source directory.
<code>path_new</code>	Destination directory.
<code>.dry_run</code>	Whether to preview the sync plan without changing files. Defaults to <code>TRUE</code> .
<code>.delete</code>	Whether to delete files in <code>path_new</code> that are absent from <code>path</code> . Defaults to <code>FALSE</code> , so extra destination files are ignored unless deletion is explicitly enabled.
<code>.compare</code>	Method used to determine whether files differ. <code>"metadata"</code> compares file size and modification time. <code>"md5"</code> compares file contents using <code>tools::md5sum()</code> . <code>"xxhash"</code> compares file contents using <code>xxhashlite::xxhash_file()</code> , if the <code>xxhashlite</code> package is installed.

Details

By default, this function performs a dry run: it prints a summary of the sync plan and returns the plan invisibly without changing any files. Set `.dry_run = FALSE` to copy, overwrite, or delete files.

The possible values of `action` are:

- `"copy"`: copy a file from `path` because it is missing from `path_new`;
- `"overwrite"`: overwrite a destination file because the source and destination files differ;
- `"delete"`: delete an extra destination file, when `.delete = TRUE`;
- `"ignore"`: leave an extra destination file unchanged, when `.delete = FALSE`;
- `"none"`: leave a file unchanged.

With `.compare = "metadata"`, files are treated as different when their size or modification time differs. With `.compare = "md5"` or `.compare = "xxhash"`, files that exist on both sides are compared by content hash instead. Under hash comparison, files with the same contents are treated as unchanged even if their modification times differ.

Value

A data frame describing the directory sync plan, invisibly. The plan includes the relative file path, the planned **action**, the **reason** for the action, source and destination file metadata, and the source and destination paths used for copying.

Examples

```
dir_from <- tempfile()
dir_to <- tempfile()

fs::dir_create(dir_from)
fs::dir_create(dir_to)

writeLines("same", fs::path(dir_from, "same.txt"))
writeLines("same", fs::path(dir_to, "same.txt"))

writeLines("copy me", fs::path(dir_from, "new.txt"))

writeLines("new contents", fs::path(dir_from, "changed.txt"))
writeLines("old contents", fs::path(dir_to, "changed.txt"))

writeLines("extra", fs::path(dir_to, "extra.txt"))

# Preview the sync plan without changing files
plan <- dir_sync_down(dir_from, dir_to)

# Actually copy and overwrite files, but keep extra destination files
dir_sync_down(dir_from, dir_to, .dry_run = FALSE)

# Also delete extra destination files
dir_sync_down(dir_from, dir_to, .dry_run = FALSE, .delete = TRUE)
```

```
extract_function_source
```

Extract a function's source and preceding comments

Description

`extract_function_source()` returns the original source lines for a function, including the contiguous block of comment lines immediately above it. Formatting, indentation, and comments inside the function are preserved.

Usage

```
extract_function_source(fun)
```

Arguments

fun A function with a `srcref` attribute whose `srcfile` contains the original source lines. Use `keep.source = TRUE` when calling `base::source()` or `base::parse()` to retain this information.

Details

A comment line starts with `#`, optionally preceded by whitespace. This includes roxygen comments. A blank line or a line of code ends the preceding comment block.

Complete lines are returned, so an assignment such as `f <-` on the same line as `function` is included, as is any other text on the first or last line. An assignment on an earlier line is not included.

This function requires retained source text; it does not reconstruct code from the function body. Functions without source references, including many functions from installed packages, cannot be used.

Value

A character vector with one element per source line, starting with any immediately preceding comments. An empty string is appended unless the last element is already empty, so concatenated results have a blank line between functions.

Supplemental materials

The main use case is printing function definitions later in our supplemental materials documents. This lets us curate which analysis functions are shown and their ordering while preserving their original comments and formatting.

In a knitr document, collect the source in a hidden chunk, then display it in a later chunk using the `code` option:

```
```{r, include = FALSE}
If not already source()-ed:
source("R/functions.R", keep.source = TRUE)
l <- c(
 extract_function_source(some_analysis_function),
 extract_function_source(some_other_analysis_function)
)
```

```{r, code = l}
```
```

Examples

```
code <- c(
  "# Add one to a number",
  "add_one <- function(x) {",
  "  x + 1",
  "}"
```

```

    "}"
  )
  env <- new.env()
  eval(parse(text = code, keep.source = TRUE), envir = env)
  extract_function_source(env$add_one)
  writeLines(extract_function_source(env$add_one))

```

| | |
|-------------------|--|
| file_replace_name | <i>Rename file basenames using functions</i> |
|-------------------|--|

Description

file_replace_name() uses `stringr::str_replace()` to rename files. file_rename_with() allows you to rename files with a generic string-transforming function.

Usage

```

file_replace_name(
  path,
  pattern,
  replacement,
  .dry_run = TRUE,
  .overwrite = FALSE
)

file_rename_with(path, .fn, ..., .dry_run = TRUE, .overwrite = FALSE)

```

Arguments

| | |
|----------------------|--|
| path | vector of paths for files to rename |
| pattern, replacement | arguments forwarded to <code>stringr::str_replace()</code> |
| .dry_run | when FALSE, files are renamed. When TRUE (the default), no files are renamed but the rename plan is printed. |
| .overwrite | Whether to overwrite files. Defaults to FALSE so that overwriting files is opt-in. |
| .fn | function to apply to file basenames |
| ... | arguments passed onto .fn |

Details

Only the basename of the file (returned by `basename()`) undergoes string replacement.

Value

a dataframe describing the file-renaming plan. Files that would overwrite an existing file are skipped unless `.overwrite = TRUE`. This function throws an error if a name collision is detected, where two files are both renamed into the same target path.

Examples

```
# With .dry_run = TRUE, we can make up some file paths.
dir <- "//some-fake-location/"
path <- file.path(
  dir,
  c("report_1.csv", "report_2.csv", "report-1.csv", "skipped.csv")
)

updated <- file_replace_name(path, "report_", "report-", .dry_run = TRUE)

# Collisions are detected
updated <- file_replace_name(path, "report_\\d", "report-1", .dry_run = TRUE)

# Doing nothing
updated <- file_rename_with(path, identity, .dry_run = TRUE)
```

| | |
|-----------------|--|
| fit_beta_gamlss | <i>Fit a beta regression model (for intelligibility)</i> |
|-----------------|--|

Description

The function fits the same type of GAMLSS model as used in [Hustad and colleagues \(2021\)](#): A beta regression model (via `gamlss.dist::BE()`) with natural cubic splines on the mean (μ) and scale (σ). This model is fitted using this package's `mem_gamlss()` wrapper function.

Usage

```
fit_beta_gamlss(data, var_x, var_y, df_mu = 3, df_sigma = 2, control = NULL)

fit_beta_gamlss_se(
  data,
  name_x,
  name_y,
  df_mu = 3,
  df_sigma = 2,
  control = NULL
)

predict_beta_gamlss(newdata, model, centiles = c(5, 10, 50, 90, 95))

optimize_beta_gamlss_slope(
  model,
  centiles = 50,
  interval = c(30, 119),
  maximum = TRUE
)

uniroot_beta_gamlss(model, centiles = 50, targets = 0.5, interval = c(30, 119))
```

Arguments

| | |
|------------------------------|---|
| <code>data</code> | a data frame |
| <code>var_x, var_y</code> | (unquoted) variable names giving the predictor variable (e.g., <code>age</code>) and outcome variable (e.g., <code>intelligibility</code>). |
| <code>df_mu, df_sigma</code> | degrees of freedom. If 0 is used, the <code>splines::ns()</code> term is dropped from the model formula for the parameter. |
| <code>control</code> | a <code>gamlss::gamlss.control()</code> controller. Defaults to NULL which uses default settings, except for setting <code>trace</code> to <code>FALSE</code> to silence the output from <code>gamlss</code> . |
| <code>name_x, name_y</code> | quoted variable names giving the predictor variable (e.g., <code>"age"</code>) and outcome variable (e.g., <code>"intelligibility"</code>). These arguments apply to <code>fit_beta_gamlss_se()</code> . |
| <code>newdata</code> | a one-column dataframe for predictions |
| <code>model</code> | a model fitted by <code>fit_beta_gamlss()</code> |
| <code>centiles</code> | centiles to use for prediction. Defaults to <code>c(5, 10, 50, 90, 95)</code> for <code>predict_beta_gamlss()</code> . Defaults to 50 for <code>optimize_beta_gamlss_slope()</code> and <code>uniroot_beta_gamlss()</code> , although both of these functions support multiple centile values. |
| <code>interval</code> | for <code>optimize_beta_gamlss_slope()</code> , the range of <code>x</code> values to optimize over. For <code>uniroot_beta_gamlss()</code> , the range of <code>x</code> values to search for roots (target <code>y</code> values) in. |
| <code>maximum</code> | for <code>optimize_beta_gamlss_slope()</code> , whether to find the maximum slope (<code>TRUE</code>) or minimum slope (<code>FALSE</code>). |
| <code>targets</code> | for <code>uniroot_beta_gamlss()</code> , the target <code>y</code> values to use as roots. By default, <code>.5</code> is used, so that <code>uniroot_beta_gamlss()</code> returns the <code>x</code> value where the <code>y</code> value is <code>.5</code> . Multiple targets are supported. |

Details

There are two versions of this function. The main version is `fit_beta_gamlss()`, and it works with unquoted column names (e.g., `age`). The alternative version is `fit_beta_gamlss_se()`; the final "se" stands for "Standard Evaluation". This designation means that the variable names must be given as strings (so, the quoted `"age"` instead of bare name `age`). This alternative version is necessary when we fit several models using parallel computing with `furrr::future_map()` (as when using bootstrap resampling).

`predict_centiles()` will work with this function, but it will likely throw a warning message. Therefore, `predict_beta_gamlss()` provides an alternative way to compute centiles from the model. This function manually computes the centiles instead of relying on `gamlss::centiles()`. The main difference is that new `x` values go through `splines::predict.ns()` and then these are multiplied by model coefficients.

`optimize_beta_gamlss_slope()` computes the point (i.e., `age`) and rate of steepest growth for different quantiles. This function wraps over the following process:

- an internal prediction function computes a quantile at some `x` from model coefficients and spline bases.

- another internal function uses `numDeriv::grad()` to get the gradient of this prediction function for `x`.
- `optimize_beta_gamlss_slope()` uses `stats::optimize()` on the gradient function to find the `x` with the maximum or minimum slope.

`uniroot_beta_gamlss()` also uses this internal prediction function to find when a quantile growth curve crosses a given value. `stats::uniroot()` finds where a function crosses 0 (a root). If we modify our prediction function to always subtract .5 at the end, then the root for this prediction function would be the `x` value where the predicted value crosses .5. That's the idea behind how `uniroot_beta_gamlss()` works. In our work, we would use this approach to find, say, the age (root) when children in the 10th percentile (`centiles`) cross 50% intelligibility (`targets`).

GAMLSS does beta regression differently:

This part is a brief note that GAMLSS uses a different parameterization of the beta distribution for its beta family than other packages.

The canonical parameterization of the beta distribution uses shape parameters α and β and the probability density function:

$$f(y; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} y^{\alpha-1} (1-y)^{\beta-1}$$

where B is the [beta function](#).

For beta regression, the distribution is reparameterized so that there is a mean probability μ and some other parameter that represents the spread around that mean. In GAMLSS (`gamlss.dist::BE()`), they use a scale parameter σ (larger values mean more spread around mean). Everywhere else—`betareg::betareg()` and `rstanarm::stan_betareg()` in `vignette("betareg", "betareg")`, `brms::Beta()` in `vignette("brms_families", "brms")`, `mgcv::betar()`—it's a precision parameter ϕ (larger values mean more precision, less spread around mean). Here is a comparison:

$$\text{betareg, brms, mgcv, etc. } \mu = \alpha / (\alpha + \beta) \phi = \alpha + b \text{E}(y) = \mu \text{VAR}(y) = \mu(1 - \mu) / (1 + \phi)$$

$$\text{GAMLSS } \mu = \alpha / (\alpha + \beta) \sigma = (1 / (\alpha + \beta + 1)) \cdot 5 \text{E}(y) = \mu \text{VAR}(y) = \mu(1 - \mu) \sigma^2$$

Value

for `fit_beta_gamlss()` and `fit_beta_gamlss_se()`, a `mem_gamlss()`-fitted model. The `.user` data in the model includes degrees of freedom for each parameter and the `splines::ns()` basis for each parameter. For `predict_beta_gamlss()`, a dataframe containing the model predictions for `mu` and `sigma`, plus columns for each centile in `centiles`. For `optimize_beta_gamlss_slope()`, a dataframe with the optimized `x` values (maximum or minimum), the gradient at that `x` value (`objective`), and the quantile (`quantile`). For `uniroot_beta_gamlss()`, a dataframe one row per quantile/target combination with the results of calling `stats::uniroot()`. The `root` column is the `x` value where the `quantile` curve crosses the `target` value.

Source

Associated article: https://doi.org/10.1044/2021_JSLHR-21-00142

Examples

```

data_fake_intelligibility

m <- fit_beta_gamlss(
  data_fake_intelligibility,
  age_months,
  intelligibility
)

# using "qr" in summary() just to suppress a warning message
summary(m, type = "qr")

# Alternative interface
d <- data_fake_intelligibility
m2 <- fit_beta_gamlss_se(
  data = d,
  name_x = "age_months",
  name_y = "intelligibility"
)
coef(m2) == coef(m)

# how to use control to change gamlss() behavior
m_traced <- fit_beta_gamlss(
  data_fake_intelligibility,
  age_months,
  intelligibility,
  control = gamlss::gamlss.control(n.cyc = 15, trace = TRUE)
)

# The `.user` space includes the spline bases, so that we can make accurate
# predictions of new xs.
names(m$.user)

# predict logit(mean) at 55 months:
logit_mean_55 <- cbind(1, predict(m$.user$basis_mu, 55)) %*% coef(m)
logit_mean_55
stats::plogis(logit_mean_55)

# But predict_gen_gamma_gamlss() does this work for us and also provides
# centiles
new_ages <- data.frame(age_months = 48:71)
centiles <- predict_beta_gamlss(new_ages, m)
centiles

# Confirm that the manual prediction matches the automatic one
centiles[centiles$age_months == 55, "mu"]
stats::plogis(logit_mean_55)

if(requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(pivot_centiles_longer(centiles)) +
    aes(x = age_months, y = .value) +

```

```

    geom_line(aes(group = .centile, color = .centile_pair)) +
    geom_point(
      aes(y = intelligibility),
      data = subset(
        data_fake_intelligibility,
        48 <= age_months & age_months <= 71
      )
    )
  }

# Age of steepest growth for each centile
optimize_beta_gamlss_slope(
  model = m,
  centiles = c(5, 10, 50, 90),
  interval = range(data_fake_intelligibility$age_months)
)

# Manual approach: Make fine grid of predictions and find largest jump
centiles_grid <- predict_beta_gamlss(
  newdata = data.frame(age_months = seq(28, 95, length.out = 1000)),
  model = m
)
centiles_grid[which.max(diff(centiles_grid$c5)), "age_months"]

# When do children in different centiles reach 50%, 70% intelligibility?
uniroot_beta_gamlss(
  model = m,
  centiles = c(5, 10, 50),
  targets = c(.5, .7)
)

```

`fit_gen_gamma_gamlss` *Fit a generalized gamma regression model (for speaking rate)*

Description

The function fits the same type of GAMLSS model as used in [Mahr and colleagues \(2021\)](#): A generalized gamma regression model (via `gamlss.dist::GG()`) with natural cubic splines on the mean (μ), scale (σ), and shape (ν) of the distribution. This model is fitted using this package's `mem_gamlss()` wrapper function.

Usage

```

fit_gen_gamma_gamlss(
  data,
  var_x,
  var_y,
  df_mu = 3,
  df_sigma = 2,

```

```

    df_nu = 1,
    control = NULL
  )

fit_gen_gamma_gamlss_se(
  data,
  name_x,
  name_y,
  df_mu = 3,
  df_sigma = 2,
  df_nu = 1,
  control = NULL
)

predict_gen_gamma_gamlss(newdata, model, centiles = c(5, 10, 50, 90, 95))

```

Arguments

| | |
|-------------------------------------|---|
| <code>data</code> | a data frame |
| <code>var_x, var_y</code> | (unquoted) variable names giving the predictor variable (e.g., <code>age</code>) and outcome variable (e.g., <code>rate</code>). |
| <code>df_mu, df_sigma, df_nu</code> | degrees of freedom. If 0 is used, the <code>splines::ns()</code> term is dropped from the model formula for the parameter. |
| <code>control</code> | a <code>gamlss::gamlss.control()</code> controller. Defaults to <code>NULL</code> which uses default settings, except for setting <code>trace</code> to <code>FALSE</code> to silence the output from <code>gamlss</code> . |
| <code>name_x, name_y</code> | quoted variable names giving the predictor variable (e.g., <code>"age"</code>) and outcome variable (e.g., <code>"rate"</code>). These arguments apply to <code>fit_gen_gamma_gamlss_se()</code> . |
| <code>newdata</code> | a one-column dataframe for predictions |
| <code>model</code> | a model fitted by <code>fit_gen_gamma_gamlss()</code> |
| <code>centiles</code> | centiles to use for prediction. Defaults to <code>c(5, 10, 50, 90, 95)</code> . |

Details

There are two versions of this function. The main version is `fit_gen_gamma_gamlss()`, and it works with unquoted column names (e.g., `age`). The alternative version is `fit_gen_gamma_gamlss_se()`; the final "se" stands for "Standard Evaluation". This designation means that the variable names must be given as strings (so, the quoted `"age"` instead of bare name `age`). This alternative version is necessary when we fit several models using parallel computing with `furrr::future_map()` (as when using bootstrap resampling).

`predict_centiles()` will work with this function, but it will likely throw a warning message. Therefore, `predict_gen_gamma_gamlss()` provides an alternative way to compute centiles from the model. This function manually computes the centiles instead of relying on `gamlss::centiles()`. The main difference is that new x values go through `splines::predict.ns()` and then these are multiplied by model coefficients.

Value

for `fit_gen_gamma_gamlss()` and `fit_gen_gamma_gamlss_se()`, a `mem_gamlss()`-fitted model. The `.user` data in the model includes degrees of freedom for each parameter and the `splines::ns()` basis for each parameter. For `predict_gen_gamma_gamlss()`, a dataframe containing the model predictions for `mu`, `sigma`, and `nu`, plus columns for each centile in `centiles`.

Source

Associated article: https://doi.org/10.1044/2021_JSLHR-21-00206

Examples

```
data_fake_rates

m <- fit_gen_gamma_gamlss(data_fake_rates, age_months, speaking_sps)

# using "qr" in summary() just to suppress a warning message
summary(m, type = "qr")

# Alternative interface
d <- data_fake_rates
m2 <- fit_gen_gamma_gamlss_se(
  data = d,
  name_x = "age_months",
  name_y = "speaking_sps"
)
coef(m2) == coef(m)

# how to use control to change gamlss() behavior
m_traced <- fit_gen_gamma_gamlss(
  data_fake_rates,
  age_months,
  speaking_sps,
  control = gamlss::gamlss.control(n.cyc = 15, trace = TRUE)
)

# The `.user` space includes the spline bases, so that we can make accurate
# predictions of new xs.
names(m$.user)

# predict log(mean) at 55 months:
log_mean_55 <- cbind(1, predict(m$.user$basis_mu, 55)) %*% coef(m)
log_mean_55
exp(log_mean_55)

# But predict_gen_gamma_gamlss() does this work for us and also provides
# centiles
new_ages <- data.frame(age_months = 48:71)
centiles <- predict_gen_gamma_gamlss(new_ages, m)
centiles
```

```

# Confirm that the manual prediction matches the automatic one
centiles[centiles$age_months == 55, "mu"]
exp(log_mean_55)

if(requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(pivot_centiles_longer(centiles)) +
    aes(x = age_months, y = .value) +
    geom_line(aes(group = .centile, color = .centile_pair)) +
    geom_point(
      aes(y = speaking_sps),
      data = subset(
        data_fake_rates,
        48 <= age_months & age_months <= 71
      )
    )
}

# Example of 0-df splines
m <- fit_gen_gamma_gamlss(
  data_fake_rates,
  age_months,
  speaking_sps,
  df_mu = 0,
  df_sigma = 2,
  df_nu = 0
)
coef(m, what = "mu")
coef(m, what = "sigma")
coef(m, what = "nu")

# mu and nu fixed, c50 mostly locked in
predict_gen_gamma_gamlss(new_ages, m)[c(1, 9, 17, 24), ]

```

fit_kmeans

Run (scaled) k-means on a dataset.

Description

Observations are `scale()`-ed before clustering.

Usage

```
fit_kmeans(data, k, vars, args_kmeans = list())
```

Arguments

| | |
|-------------------|------------------------------|
| <code>data</code> | a dataframe |
| <code>k</code> | number of clusters to create |

vars variable selection for clustering. Select multiple variables with `c()`, e.g., `c(x, y)`. The selection supports tidyselect semantics [tidyselect::select_helpers](#), e.g., `c(x, starts_with("mean_"))`.

args_kmeans additional arguments passed to `stats::kmeans()`.

Details

Note that each variable is `scaled()` before clustering and then cluster means are unscaled to match the original data scale.

This function provides the original kmeans labels as `.kmeans_cluster` but other alternative labeling based on different sortings of the data. These are provided in order to deal with label-swapping in Bayesian models. See bootstrapping example below.

Value

the original data but augmented with additional columns for clustering details. including `.kmeans_cluster` (cluster number of each observation, as a factor) and `.kmeans_k` (selected number of clusters).

Cluster-level information is also included. For example, suppose that we cluster using the variable `x`. Then the output will have a column `.kmeans_x` giving the cluster mean for `x` and `.kmeans_rank_x` giving the cluster labels reordered using the cluster means for `x`. The column `.kmeans_sort` contains the cluster sorted using the first principal component of the scaled variables. All columns of cluster indices are a `factor()` so that they can be plotted as discrete variables.

Examples

```
data_kmeans <- fit_kmeans(mtcars, 3, c(mpg, wt, hp))

library(ggplot2)
ggplot(data_kmeans) +
  aes(x = wt, y = mpg) +
  geom_point(aes(color = .kmeans_cluster))

ggplot(data_kmeans) +
  aes(x = wt, y = mpg) +
  geom_point(aes(color = .kmeans_rank_wt))

# Example of label swapping
set.seed(123)
data_boots <- lapply(
  1:10,
  function(x) {
    rows <- sample(seq_len(nrow(mtcars)), replace = TRUE)
    data <- mtcars[rows, ]
    data$.bootstrap <- x
    data
  }
) |>
lapply(fit_kmeans, k = 3, c(mpg, wt, hp)) |>
```

```

dplyr::bind_rows() |>
dplyr::select(.bootstrap, dplyr::starts_with(".kmeans_")) |>
dplyr::distinct()

# Clusters start off in random locations and move to center, so the labels
# differ between model runs and across bootstraps.
ggplot(data_boots) +
  aes(x = .kmeans_wt, y = .kmeans_mpg) +
  geom_point(aes(color = .kmeans_cluster)) +
  labs(title = "k-means centers on 10 bootstraps")

# Labels sorted using first principal component
# so the labels are more consistent.
ggplot(data_boots) +
  aes(x = .kmeans_wt, y = .kmeans_mpg) +
  geom_point(aes(color = .kmeans_sort)) +
  labs(title = "k-means centers on 10 bootstraps")

```

format_year_month_age

Convert between age in months, years;months, and yymm age formats

Description

Convert between age in months, years;months, and yymm age formats

Usage

```
format_year_month_age(x, sep = ";")
```

```
parse_year_month_age(x, sep = ";")
```

```
parse_yymm_age(x, start = 1L)
```

Arguments

- | | |
|--------------|--|
| x | <ul style="list-style-type: none"> • <code>format_year_month_age()</code>: a numeric vector of (non-negative) ages in months. • <code>parse_year_month_age()</code>: a character vector of ages in "years;months" format (or <code>years{sep}months</code> format more generally). • <code>parse_yymm_age()</code>: a character vector of ages in "yy-mm" format. |
| sep | Separator to use for <code>year_month</code> functions. Defaults to <code>;</code> . |
| start | For <code>parse_yymm_age()</code> , the location of the starting character the <code>yymm</code> sequence. Defaults to 1. |

Details

For `format_year_month_age()`, ages of NA return "NA;NA".

For `parse_year_month_age()`, values that cannot be parsed return NA.

This format by default is not numerically ordered. This means that `c("2;0", "10;10", "10;9")` would sort as `c("10;10", "10;9", "2;0")`. The function `stringr::str_sort(..., numeric = TRUE)` will sort this vector correctly.

Value

- `format_year_month_age()` returns a character vector in "years;months" format (or years{sep}months format more generally).
- `parse_year_month_age()` returns a vector of ages in months.
- `parse_yymm_age()`: returns a vector of ages in months.

Examples

```
ages <- c(26, 58, 25, 67, 21, 59, 36, 43, 27, 49, NA)
ym_ages <- format_year_month_age(ages)
ym_ages

parse_year_month_age(ym_ages)

parse_yymm_age(c("0204", "0310"))

parse_yymm_age(c("ab_0204", "ab_0310"), start = 4)
```

impute_values_by_length

Staged imputation

Description

Impute missing data at different utterance lengths using successive linear models.

Usage

```
impute_values_by_length(
  data,
  var_y,
  var_length,
  id_cols = NULL,
  include_max_length = FALSE,
  data_train = NULL
)
```

Arguments

| | |
|---------------------------|---|
| data | dataframe in which to impute missing value |
| var_y | bare name of the response variable for imputation |
| var_length | bare name of the length variable |
| id_cols | a selection of variable names that uniquely identify each group of related observations. For example, <code>c(child_id, age_months)</code> . |
| include_max_length | whether to use the maximum length value as a predictor in the imputation models. Defaults to FALSE . |
| data_train | (optional) dataframe used to train the imputation models. For example, we might have data from a reference group of children in <code>data_train</code> but a clinical population in <code>data</code> . If omitted, the dataframe in <code>data</code> is used to train the models. <code>data_train</code> can also be a function. In this case, it is applied to the <code>data</code> argument in order to derive (filter) a subset of the data for training. |

Value

a dataframe with the additional columns `{var_y}_imputed` (the imputed value), `.max_{var_length}` with the highest value of `var_length` with observed data, and `{var_y}_imputation` for labeling whether observations were "imputed" or "observed".

Background

In Hustad and colleagues (2020), we modeled intelligibility data in young children's speech. Children would hear an utterance and then they would repeat it. The utterances started at 2 words in length, then increased to 3 words in length, and so on in batches of 10 sentences, all the way to 7 words in length. There was a problem, however: Not all of the children could produce utterances at every length. Specifically, if a child could not reliably produced 5 utterances of a given length length, the task was halted. So given the nature of the task, if a child had produced 5-word utterances, they also produced 2–4-word utterances as well.

The length of the utterance probably influenced the outcome variable: Longer utterances have more words that might help a listener understand the sentence, for example. Therefore, it did not seem appropriate to ignore the missing values. We used the following two-step procedure (see the [Supplemental Materials](#) for more detail):

Other notes:

Remark about `data` and `data_train`: One might ask, *why shouldn't some children help train the data imputation models?* Let's consider a norm-referenced standardized testing scenario: We have a new participant (observations in `data`), and we want to know how they compare to their age peers (participants in `data_train`). By separating out `data_train` and fixing it to a reference group, we can apply the same adjustment/imputation procedure to all new participants.

References

- Hustad, K. C., Mahr, T., Natzke, P. E. M., & Rathouz, P. J. (2020). Development of Speech Intelligibility Between 30 and 47 Months in Typically Developing Children: A Cross-Sectional Study of Growth. *Journal of Speech, Language, and Hearing Research*, 63(6), 1675–1687. https://doi.org/10.1044/2020_JSLHR-20-00008
- Hustad, K. C., Mahr, T., Natzke, P. E. M., & J. Rathouz, P. (2020). Supplemental Material S1 (Hustad et al., 2020). ASHA journals. <https://doi.org/10.23641/asha.12330956.v1>

Examples

```
set.seed(1)
fake_data <- tibble::tibble(
  child = c(
    "a", "a", "a", "a", "a",
    "b", "b", "b", "b", "b",
    "c", "c", "c", "c", "c",
    "e", "e", "e", "e", "e",
    "f", "f", "f", "f", "f",
    "g", "g", "g", "g", "g",
    "h", "h", "h", "h", "h",
    "i", "i", "i", "i", "i"
  ),
  level = c(1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5),
  x = c(
    c(100, 110, 120, 130, 150) + c(-8, -5, 0, NA, NA),
    c(100, 110, 120, 130, 150) + c(6, 6, 4, NA, NA),
    c(100, 110, 120, 130, 150) + c(-5, -5, -2, 2, NA),
    c(100, 110, 120, 130, 150) + rbinom(5, 12, .5) - 6,
    c(100, 110, 120, 130, 150) + rbinom(5, 12, .5) - 6,
    c(100, 110, 120, 130, 150) + rbinom(5, 12, .5) - 6,
    c(100, 110, 120, 130, 150) + rbinom(5, 12, .5) - 6,
    c(100, 110, 120, 130, 150) + rbinom(5, 12, .5) - 6
  )
)
data_imputed <- impute_values_by_length(
  fake_data,
  x,
  level,
  id_cols = c(child),
  include_max_length = FALSE
)

if (requireNamespace("ggplot2")) {
  library(ggplot2)
  ggplot(data_imputed) +
    aes(x = level, y = x_imputed) +
    geom_line(aes(group = child)) +
    geom_point(aes(color = x_imputation))
}
```

| | |
|----------------|--------------------------------------|
| info_surprisal | Compute entropy and related measures |
|----------------|--------------------------------------|

Description

Compute entropy and related measures

Usage

```
info_surprisal(x, base = NULL)

info_entropy(p, base = NULL)

info_cross_entropy(p, q, base = NULL)

info_kl_divergence(p, q, base = NULL)

info_kl_divergence_matrix(mat, base = NULL)
```

Arguments

| | |
|-------------|---|
| x | a vector of probabilities |
| base | the base of the logarithm. Defaults to e (<code>exp(1)</code>). |
| p, q | a probability distribution (vector of probabilities that sum to 1) |
| mat | a matrix where each row is a probability distribution |

Value

the entropy

Information theory basics

Given a probability **x**, the **surprisal value** (or information content) of **x** is the log of the inverse probability, $\log(1 / x)$. Rare events (smaller probabilities) have larger surprisal values than more common events. The word "surprise" here conveys how unexpected or informative an event is. (The idea that surprises are "informative" or contain information makes sense with the intuition that there isn't much to be learned from predictable events.) Because of how division works with logarithms, $\log(1 / x)$ simplifies so that `info_surprisal(x)` is the negative log probability, $-\log(x)$. The units of the information content depends on the base of the logarithm. Our functions by default use the natural `log()` which provides information in *nats*, but it is also common to see `log2()`-based surprisals which provide information in *bits*.

Given a probability distribution **p**—a vector of probabilities that sum to 1—the **entropy** of the distribution is the probability-weighted average of the surprisal values. A weighted average is `sum(weights * values) / sum(weights)`, but because the weights here are probabilities that sum to 1, the `info_entropy(p)` is `sum(p * info_surprisal(p))`. Entropy

can be interpreted as a measure of uncertainty in the distribution; it's the expected surprisal value in the distribution.

In `info_entropy(p)` the surprisal values and their weights came from the same probability distribution. But this doesn't need to be the case. Suppose that there are ground-truth probabilities `p` and there are also estimated probabilities `q`. `info_entropy(q)` computes a weighted average where events with surprisal `info_surprisal(q)` have frequencies of `q`. But if we knew the true frequencies, the surprisals in `q` would occur with frequency `p`, so their weighted average should be `sum(p * info_surprisal(q))`. This value is the **cross entropy** of `q` with respect to `p`, and `info_cross_entropy(p, q)` implements this function. Cross entropy is commonly used as a loss function in machine learning.

Gibb's inequality says that `info_entropy(p) <= info_cross_entropy(p, q)`. Unless `p` and `q` are the same distribution, there will always be some excess uncertainty or surprise in the cross entropy compared to the entropy: `info_cross_entropy(p, q) = info_entropy(p) + *excess*`. This excess is the **Kullback-Liebler divergence** (KL divergence or relative entropy). Due to the properties of logarithms, `info_kl_divergence(p, q)` simplifies to `sum(p * log(p / q))`. KL divergence is an important quantity for comparing probability distributions. For example, each row in a confusion matrix is a probability distribution, and KL divergence can identify which rows are most similar. `info_kl_divergence_matrix(mat)` computes a distance matrix using on each pair of rows in `mat` using KL divergence.

Examples

```
wikipedia_example <- rbind(
  p = c(9, 12, 4) / 25,
  q = c(1, 1, 1) / 3
)

info_kl_divergence_matrix(wikipedia_example)
```

| | |
|---------------|-------------------------------------|
| join_to_split | <i>Join data onto resampled IDs</i> |
|---------------|-------------------------------------|

Description

Join data onto resampled IDs

Usage

```
join_to_split(x, y, by, validate = FALSE)
```

Arguments

| | |
|-----------------------|---|
| <code>x</code> | an rset object created by <code>rsample::bootstraps()</code> |
| <code>y</code> | y dataframe with a column of the id values which was resampled to create <code>x</code> |
| <code>by</code> | the name of column in <code>y</code> with the data |
| <code>validate</code> | whether to validate the join by counting the number of rows associated with each id. Defaults to <code>FALSE</code> . |

Value

the original rset object with its `x$data` updated to join with `y` and with the row numbers `x$in_id` updated to work on the expanded dataset.

Examples

```
library(dplyr)
data_trees <- tibble::as_tibble(datasets::Orange)

data_tree_ids <- distinct(data_trees, Tree)

# Resample ids
data_bootstraps <- data_tree_ids %>%
  rsample::bootstraps(times = 20) %>%
  rename(splits_id = splits) %>%
  # Attach data to resampled ids
  mutate(
    data_splits = splits_id %>% purrr::map(
      join_to_split,
      data_trees,
      by = "Tree",
      validate = TRUE
    )
  )

data_bootstraps
```

| | |
|----------------|---|
| logitnorm_mean | <i>Compute the mean of logit-normal distribution(s)</i> |
|----------------|---|

Description

This function is a wrapper around `logitnorm::momentsLogitnorm()`.

Usage

```
logitnorm_mean(mu, sigma)
```

Arguments

| | |
|--------------------|--|
| <code>mu</code> | mean(s) on the logit scale |
| <code>sigma</code> | standard deviation(s) on the logit scale |

Value

the means of the distributions

Examples

```
x <- logitnorm_mean(2, 1)
x

# compare to simulation
set.seed(100)
rnorm(1000, 2, 1) |> plogis() |> mean()
```

| | |
|------------|---|
| mem_gamlss | <i>Fit a gamlss model but store user data</i> |
|------------|---|

Description

Think of it as a gamlss model with memories (mem. gamlss).

Usage

```
mem_gamlss(...)
```

Arguments

... arguments passed to `gamlss::gamlss()`

Value

the fitted `model` object but updated to include user information in `model$.user`. Includes the dataset used to fit the model `model$.user$data`, the session info `model$.user$session_info` and the call used to fit the model `model$.user$call`. `model$call` is updated to match

| | |
|------------------|--|
| predict_centiles | <i>Predict and tidy centiles from a GAMLSS model</i> |
|------------------|--|

Description

`gamlss` has trouble doing predictions without the original training data.

Usage

```
predict_centiles(newdata, model, centiles = c(5, 10, 50, 90, 95), ...)

pivot_centiles_longer(data, prefix = "c")
```

Arguments

| | |
|-----------------------|--|
| <code>newdata</code> | a one-column dataframe for predictions |
| <code>model</code> | a gamlss model prepared by <code>mem_gamlss()</code> |
| <code>centiles</code> | centiles to use for prediction. Defaults to <code>c(5, 10, 50, 90, 95)</code> . |
| <code>...</code> | arguments passed to <code>gamlss::centiles.pred()</code> |
| <code>data</code> | centile predictions to reshape for <code>pivot_centiles_longer()</code> |
| <code>prefix</code> | String giving the prefix shared by centile columns. Defaults to <code>"c"</code> . |

Value

a tibble with fitted centiles for `predict_centiles()` and a long-format tibble with one centile value per row in `pivot_centiles_longer()`

| | |
|---------------------------|---|
| <code>print_duckdb</code> | <i>Custom function for printing duckdb database connections</i> |
|---------------------------|---|

Description

Custom function for printing duckdb database connections

Usage

```
print_duckdb(object, table_limit = 50)
```

Arguments

| | |
|--------------------------|---|
| <code>object</code> | a database connection to a duckdb database |
| <code>table_limit</code> | number of table names to print before truncating with <code>....</code> . Defaults to 50. |

Details

Use the following to overwrite the S4 method for printing duckdb objects.

```
setMethod("show", "duckdb_connection", wisclabmisc::print_duckdb)
```

Value

NULL invisibly

skip_block*Skip a block of code without executing it*

Description

skip_block() is a lightweight control-flow helper to deliberately skip execution of a block of code while still documenting that the block exists. It is intended as an alternative to commenting out code or wrapping code in `if (FALSE) { ... }`.

Usage

```
skip_block(...)
```

Arguments

... Either a single braced expression containing the code block to skip, or a character string followed by a single braced expression. When a message is supplied, it is printed instead of the default message.

Details

The function captures the unevaluated code block, reports how many lines would have run, optionally prints a user-supplied message, and then returns invisibly without evaluating the code.

skip_block() uses non-standard evaluation to capture the code block via **substitute()**. The code block is never evaluated.

Supplying more than two arguments, or supplying no code block, is an error.

Value

NULL, invisibly.

Examples

```
skip_block({
  Sys.sleep(10)
})

skip_block("Skipping slow preprocessing step", {
  Sys.sleep(10)
})
```

| | |
|----------|---|
| tbl_bind | <i>Select and row-bind multiple database tables from the same source together</i> |
|----------|---|

Description

Select and row-bind multiple database tables from the same source together

Usage

```
tbl_bind(src, table_selection, id_name = ".source")
```

Arguments

| | |
|-----------------|--|
| src | a database connection |
| table_selection | a tidy-select name selection. For example, <code>starts_with("loc_")</code> would select all tables that start with "loc_". Place multiple selections inside of <code>c()</code> . |
| id_name | Name for the table ID column. The names of the original tables will be included in a column named by <code>id_name</code> . By default, this column is <code>".source"</code> . If <code>NULL</code> , then the ID column is not included. |

Details

In `dplyr`, `tbl(src, table_name)` names and queries a database table. `tbl_bind(src, <selection>)` extends this idea into selecting and combining multiple database tables.

To manually combine two `tbls` `x` and `y`, use `dplyr::union_all(x, y)`.

Value

a `tbl` (query to a remote table)

Examples

```
library(dplyr)
db <- duckdb::duckdb() |>
  DBI::dbConnect()
db

DBI::dbWriteTable(db, "mtcars_g1", mtcars[mtcars$cyl == 4, ])
DBI::dbWriteTable(db, "mtcars_g2", mtcars[mtcars$cyl == 6, ])
DBI::dbWriteTable(db, "mtcars_g3", mtcars[mtcars$cyl == 8, ])
DBI::dbWriteTable(db, "trees", trees)

DBI::dbListTables(db)

r <- db |>
```

```
tbl_bind(starts_with("mtcars")) |>
  count(.source)
r

# the query is several UNIONs
show_query(r)

DBI::dbDisconnect(db, shutdown = TRUE)
```

tocs_item

Extract the TOCS details from a string (usually a filename)

Description

Extract the TOCS details from a string (usually a filename)

Usage

```
tocs_item(xs)

tocs_type(xs)

tocs_length(xs)
```

Arguments

xs a character vector

Value

`tocs_item()` returns the substring with the TOCS item, `tocs_type()` returns whether the item is "single-word" or "multiword", and `tocs_length()` returns the length of the TOCS item (i.e., the number of words).

Examples

```
x <- c(
  "XXv16s7T06.lab", "XXv15s5T06.TextGrid", "XXv13s3T10.WAV",
  "XXv18wT11.wav", "non-matching", "s2T01",
  "XXv01s4B01.wav", "XXv01wB01.wav",
  # sometimes these have tags for *v*irtual visits or recording attempts
  "XXv13s3T10v.WAV", "XXv13s3T10a.lab"
)
data.frame(
  x = x,
  item = tocs_item(x),
  type = tocs_type(x),
  length = tocs_length(x)
)
```

| | |
|---------------|--|
| trapezoid_auc | <i>Compute AUCs using the trapezoid method</i> |
|---------------|--|

Description

Compute AUCs using the trapezoid method

Usage

```
trapezoid_auc(xs, ys)

partial_trapezoid_auc(xs, ys, xlim)
```

Arguments

| | |
|---------------------|---|
| <code>xs, ys</code> | x and y positions |
| <code>xlim</code> | two-element vector (a range) of the <code>xs</code> to sum over |

Value

the area under the curve computed using the trapezoid method. For `partial_trapezoid_auc()`, the partial area under the curve is computed.

Examples

```
# Predict whether a car has automatic vs. manual transmission from mpg
r <- pROC::roc(am ~ mpg, mtcars)
pROC::auc(r)
trapezoid_auc(r$specificities, r$sensitivities)

pROC::auc(r, partial.auc = c(.9, 1), partial.auc.focus = "sp")
partial_trapezoid_auc(r$specificities, r$sensitivities, c(.9, 1))

pROC::auc(r, partial.auc = c(.9, 1), partial.auc.focus = "se")
partial_trapezoid_auc(r$sensitivities, r$specificities, c(.9, 1))

pROC::auc(r, partial.auc = c(.1, .9), partial.auc.focus = "sp")
partial_trapezoid_auc(r$specificities, r$sensitivities, c(.1, .9))

pROC::auc(r, partial.auc = c(.1, .9), partial.auc.focus = "se")
partial_trapezoid_auc(r$sensitivities, r$specificities, c(.1, .9))
```

weight_lengths_with_ordinal_model

Weight utterance lengths by using an ordinal regression model

Description

For each participant, we find their length of longest utterance. We predict this longest utterance length as a nonlinear function of some variable, and we compute the probability of reaching each utterance length at each value of the predictor variable. These probabilities are then normalized to provide weights for each utterance length.

Usage

```
weight_lengths_with_ordinal_model(
  data_train,
  var_length,
  var_x,
  id_cols,
  spline_df = 2,
  data_join = NULL
)
```

Arguments

| | |
|-------------------|--|
| data_train | dataframe used to train the ordinal model. data_train can also be a function. In this case, it is applied to the data_join argument in order to derive (filter) a subset of the data for training. |
| var_length | bare name of the length variable. For example, tocs_level . |
| var_x | bare name of the predictor variable. For example, age_months . |
| id_cols | a selection of variable names that uniquely identify each group of related observations. For example, c(child_id, age_months) . |
| spline_df | number of degrees of freedom to use for the ordinal regression model. |
| data_join | (optional) dataset to use join the weights onto. This feature is necessary because we want to train a dataset on the observed data but supply the weights to the dataset with missing values imputed. |

Value

the probability and weights of each utterance length at each observed value of **var_x**. These are in the added columns **{var_length}_prob_reached** and **{var_length}_weight**, respectively.

References

- Hustad, K. C., Mahr, T., Natzke, P. E. M., & Rathouz, P. J. (2020). Development of Speech Intelligibility Between 30 and 47 Months in Typically Developing Children: A Cross-Sectional Study of Growth. *Journal of Speech, Language, and Hearing Research*, 63(6), 1675–1687. https://doi.org/10.1044/2020_JSLHR-20-00008
- Hustad, K. C., Mahr, T., Natzke, P. E. M., & J. Rathouz, P. (2020). Supplemental Material S1 (Hustad et al., 2020). ASHA journals. <https://doi.org/10.23641/asha.12330956.v1>

Examples

```
data_weights <- weight_lengths_with_ordinal_model(
  data_example_intelligibility_by_length,
  tocs_level,
  age_months,
  child,
  spline_df = 2
)

if (requireNamespace("ggplot2")) {
  library(ggplot2)
  p1 <- ggplot(data_weights) +
    aes(x = age_months, y = tocs_level_prob_reached) +
    geom_line(aes(color = ordered(tocs_level)), linewidth = 1) +
    scale_color_ordinal(end = .85) +
    labs(y = "Prob. of reaching length", color = "Utterance length")
  print(p1)

  p2 <- p1 +
    aes(y = tocs_level_weight) +
    labs(y = "Weight of utterance length")
  print(p2)
}
```

Index

- * **data-utils**
 - chrono_age, 6
 - format_year_month_age, 46
 - impute_values_by_length, 47
 - tocs_item, 57
 - weight_lengths_with_ordinal_model, 59
- * **database**
 - print_duckdb, 54
 - tbl_bind, 56
- * **datasets**
 - data_acq_consonants, 14
 - data_example_intelligibility_by_length, 23
 - data_fake_intelligibility, 24
 - data_fake_rates, 25
 - data_features_consonants, 25
 - data_tocs_items, 29
- * **etc**
 - join_to_split, 51
- * **gamlss**
 - check_model_centiles, 5
 - mem_gamlss, 53
 - predict_centiles, 53
- * **models**
 - fit_beta_gamlss, 37
 - fit_gen_gamma_gamlss, 41
- * **other-stats**
 - fit_kmeans, 44
 - info_surprisal, 50
 - logitnorm_mean, 52
- * **programming-utils**
 - audit_wrap, 2
 - extract_function_source, 34
 - skip_block, 55
- * **roc**
 - compute_empirical_roc, 7
 - compute_predictive_value_from_rates, 10
 - compute_sens_spec_from_ecdf, 11
 - compute_smooth_density_roc, 13
 - trapezoid_auc, 58
- audit_peek (*audit_wrap*), 2
- audit_poke (*audit_wrap*), 2
- audit_unwrap (*audit_wrap*), 2
- audit_wrap, 2
- base::parse(), 35
- base::source(), 35
- basename(), 36
- betareg::betareg(), 39
- brms::Beta(), 39
- brms::brm(), 4, 5
- brms_args_create, 4
- check_computed_centiles
(*check_model_centiles*), 5
- check_model_centiles, 5
- chrono_age, 6
- compute_empirical_roc, 7
- compute_overlap_rate, 9
- compute_predictive_value_from_rates, 10
- compute_sens_spec_from_ecdf, 11
- compute_smooth_density_roc, 13
- data_acq_consonants, 14, 25
- data_acq_vowels
(*data_acq_consonants*), 14
- data_example_intelligibility_by_length, 23
- data_fake_intelligibility, 24
- data_fake_rates, 25
- data_features_consonants, 14, 25
- data_features_vowels
(*data_features_consonants*), 25
- data_tocs_items, 29

dir_sync_down, 32
 extract_function_source, 34
 file_rename_with (*file_replace_name*), 36
 file_replace_name, 36
 fit_beta_gamlss, 37
 fit_beta_gamlss(), 38
 fit_beta_gamlss_se (*fit_beta_gamlss*), 37
 fit_gen_gamma_gamlss, 41
 fit_gen_gamma_gamlss(), 42
 fit_gen_gamma_gamlss_se (*fit_gen_gamma_gamlss*), 41
 fit_kmeans, 44
 format_year_month_age, 46
 furrr::future_map(), 38, 42

 gamlss.dist::BE(), 37, 39
 gamlss.dist::GG(), 41
 gamlss::centiles(), 38, 42
 gamlss::gamlss.control(), 38, 42

 impute_values_by_length, 47
 info_cross_entropy (*info_surprisal*), 50
 info_entropy (*info_surprisal*), 50
 info_kl_divergence (*info_surprisal*), 50
 info_kl_divergence_matrix (*info_surprisal*), 50
 info_surprisal, 50

 join_to_split, 51

 logitnorm::momentsLogitnorm(), 52
 logitnorm_mean, 52

 mem_gamlss, 53
 mem_gamlss(), 37, 39, 41, 43
 mgcv::betar(), 39

 optimize_beta_gamlss_slope (*fit_beta_gamlss*), 37

 parse_year_month_age (*format_year_month_age*), 46
 parse_yymm_age (*format_year_month_age*), 46

 partial_trapezoid_auc (*trapezoid_auc*), 58
 pivot_centiles_longer (*predict_centiles*), 53
 predict_beta_gamlss (*fit_beta_gamlss*), 37
 predict_centiles, 53
 predict_centiles(), 38, 42
 predict_gen_gamma_gamlss (*fit_gen_gamma_gamlss*), 41
 print_duckdb, 54

 rstanarm::stan_betareg(), 39

 skip_block, 55
 splines::ns(), 38, 39, 42, 43
 splines::predict.ns(), 38, 42
 stats::uniroot(), 39
 stringr::str_replace(), 36

 tbl_bind, 56
 tidysselect::select_helpers, 45
 tocs_item, 57
 tocs_length (*tocs_item*), 57
 tocs_type (*tocs_item*), 57
 tools::md5sum(), 33
 trapezoid_auc, 58

 uniroot_beta_gamlss (*fit_beta_gamlss*), 37

 weight_lengths_with_ordinal_model, 59