Package 'XSLmodels'

November 10, 2026

Title Run and Optimize Cross-Situational Word Learning Models

Version 0.5.0

Maintainer George Kachergis <george.kachergis@gmail.com>

Description A collection of cognitive models of cross-situational word learning, together with methods and datasets for fitting and cross-validating them against experimental and corpus data, and for plotting the results.

URL https://www.kachergis.com/XSLmodels/, https://github.com/kachergis/XSLmodels

BugReports https://github.com/kachergis/XSLmodels/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports DEoptim, dplyr, ggplot2, purrr, rlang, tibble, tidyr

Suggests gganimate, knitr, rmarkdown, testthat (>= 3.0.0), viridis

VignetteBuilder knitr

LazyData true

Config/testthat/edition 3

License GPL (>= 3)

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author George Kachergis [aut, cre] (ORCID: <https://orcid.org/0000-0003-4153-4167>), Mika Braginsky [aut] (ORCID: <https://orcid.org/0000-0001-9039-3220>), Michael C. Frank [aut] (ORCID: <https://orcid.org/0000-0002-7551-4378>, affiliation: Stanford University)

Repository CRAN

Date/Publication 2026-11-10 18:40:02 UTC

Contents

baseline . . . 3
bayesian_decay . . . 4
benitez2020 . . . 4
choice_probabilities . . . 5
create_cooc_matrix . . . 6
cross_validated_group_fits . . . 7
decay . . . 8
fazly . . . 9
fgt2009 . . . 9
fgt2009_rsa . . . 11
fgt2009_sweep_alpha . . . 12
fm_corpus . . . 13
gangwani2011_category . . . 14
get_crossvalidated_model_fit . . . 15
get_example_ambiguous_condition . . . 16
get_example_unambiguous_condition . . . 17
get_fscore . . . 17
get_group_model_fit . . . 18
get_perf . . . 19
get_roc . . . 20
get_tp . . . 21
get_train_test_split . . . 22
guess_and_test . . . 22
kachergis2012_highlighting . . . 23
kachergis_initial_accuracy . . . 24
kachergis_priming . . . 24
kalman_filter . . . 25
koehne2013 . . . 27
mafc_test . . . 28
minerva2 . . . 28
multi_sampling . . . 29
new_xslFit . . . 30
plot_training_trials . . . 31
predict_referent . . . 31
propose_but_verify . . . 33
pursuit . . . 33
rem . . . 34
rescorla_wagner . . . 35
rollins_corpus . . . 37
rsa_listener . . . 38
show_datasets . . . 39
show_models . . . 39
softmax_rl . . . 40
softmax_weights . . . 40
suanda2014 . . . 41
tilles . . . 42

todam 42
uncfam 43
uncfam_attention 44
uncfam_elimination 45
uncfam_gamma 46
uncfam_general 47
uncfam_predictive 49
uncfam_sampling 50
update_params 50
vlach_debrock2017 51
xslControl-class 52
xslData-class 53
xslFit-class 54
xsl_datasets 55
xsl_fit 55
xsl_holdout_datasets 57
xsl_run 58

Index **59**

`baseline` *Baseline model*

Description

Baseline model

Usage

```
baseline()
```

Value

An object of class xslMod

Examples

```
mod <- baseline()
xsl_run(mod, get_example_ambiguous_condition())
```

bayesian_decay *Bayesian decay model*

Description

Bayesian decay model

Usage

```
bayesian_decay(alpha, delta, ch_dec)
```

Arguments

alpha	Decay for word/object non-co-occurrence (0.1, 0.5, 0.9)
delta	Multiplier for word/object co-occurrence (1 for no increase)
ch_dec	Decision parameter (e.g., 1)

Value

An object of class xslMod

Examples

```
mod <- bayesian_decay(alpha = 0.5, delta = 1, ch_dec = 1)
xsl_run(mod, get_example_ambiguous_condition())
```

benitez2020 *Benitez et al. (2020) temporal-structure-of-naming experiment data*

Description

A named list of 7 xslData objects from Benitez, V. L., et al. (2020), "The temporal structure of naming events differentially affects children's and adults' cross-situational word learning" (data/materials: https://osf.io/2hmxr/): one per condition (Interleaved, Massed, Unstructured) x age-group (kids, adults) combination, except Unstructured's first training order, which only has child data. Unstructured has two training orders; its second and third were confirmed identical by the original extraction and are pooled as "order 2/3".

Usage

```
benitez2020
```

Format

An object of class `list` of length 7.

Details

8 symmetric word-object pairs (word `i` pairs with object `i`), 2 pairs per training trial, 12 trials, followed by an 8-item 2AFC test. `accuracy` and `response_matrix` are real per-word 2AFC accuracy and raw response counts for that condition/age-group combination (not an estimate) – structurally this could join xsl_datasets, but is kept standalone deliberately as held-out generalization-test data; see `xsl_holdout_datasets`. `data-raw/Benitez2020-rep/Benitez2020_XSLdata_extraction.R` extracts the raw OSF data into training orders, 2AFC test trials, and response matrices; `data-raw/add_benitez2020.R` builds the `xslData` objects from those.

Examples

```
benitez2020[["Interleaved, kids"]]$accuracy
m <- xsl_run(baseline(), benitez2020[["Massed, adults"]])$fits[[1]]$matrix
mafc_test(m, benitez2020[["Massed, adults"]]$test)
```

---

choice_probabilities *Choice probabilities for each test item, with human response counts*

Description

For each test item, returns the alternatives offered, the correct object, the model's probability of choosing each alternative, and how many participants chose each one. This is the full choice distribution behind the accuracy that `xsl_run()` scores: the probability of the correct alternative is exactly that item's entry in `perf`. It is what a multinomial likelihood (`xslControl(objective = "multinomial")`) compares with the data, and it shows which wrong objects a model expects learners to choose.

Usage

```
choice_probabilities(m, data, d = NULL, lapse = 0, block = NULL, phi = NULL)
```

Arguments

`m`	A word-by-object matrix, e.g. `xsl_run(mod, data)$fits[[1]]$matrix`.
`data`	The `xslData` condition `m` was learned from.
`d`	Optional choice exponent (a model's `ch_dec`; see `get_perf()`).
`lapse`	Lapse rate: the probability of guessing uniformly among the alternatives, mixed into `p` (0, the default, leaves `p` as the model's choice rule gives it).
`block`	For a multi-block condition, which block's responses to return (`m` should then be that block's matrix, e.g. `xsl_run(mod, data)$fits[[1]]$blocks[[block]]$matrix`).
`phi`	Optional softmax inverse temperature (a model's `phi`; see `softmax_weights()`). For a stochastic model, pass the mean matrix, `matrix / n_sim`, as `xsl_run()` scores it.

Details

Items are scored as in `xsl_run()`: on a condition with a `test`, by Luce choice over each test trial's listed objects (see `mafc_test()`); otherwise among all objects, with word i's correct referent the i-th object (see `get_perf()`).

Value

A tibble with one row per test item: `item`, `correct` (the correct object's label), and list columns `alternatives` (object labels), `p` (choice probabilities over `alternatives`) and `counts` (participants choosing each alternative; `NULL` entries if `data` has no `response_matrix`).

Examples

```
dat <- xsl_datasets[[1]]
fit <- xsl_run(uncfam(X = .1, B = 3, C = .97), dat)$fits[[1]]
ch <- choice_probabilities(fit$matrix, dat)
ch$p[[1]]       # the model's choice distribution for item 1
ch$counts[[1]]  # what participants chose
```

---

create_cooc_matrix *Creates co-occurrence matrix from training trials*

Description

Given a training order (list of words and objects per trial), this function returns a matrix tallying word-object co-occurrences across the trials. This matrix can be used to analyze the frequency with which certain words and objects appear together during the training phase.

Usage

```
create_cooc_matrix(train)
```

Arguments

`train`	A list representing the training data, where each element is a trial that includes both words and objects. The structure is expected to have sub-elements `words` and `objects` for each trial.

Value

A matrix where each element represents the count of co-occurrences between a word (rows) and an object (columns). The row and column names correspond to the unique words and objects found in the training data, respectively.

Examples

```
create_cooc_matrix(xsl_datasets[[1]]$train)
```

`cross_validated_group_fits` *Perform Cross-Validation on Group Fits of a Model*

Description

This function runs k-fold cross-validation for group fits of a specified model using a set of conditions. On each fold, the model is fit (via `xsl_fit()`) to the training conditions, and the fitted model is then evaluated (via `xsl_run()`) on the held-out test conditions to obtain an out-of-sample SSE.

Usage

```
cross_validated_group_fits(
  model,
  combined_data,
  lower,
  upper,
  n_folds = 5,
  control = xslControl(),
 deoptim_control = DEoptim::DEoptim.control(reltol = 0.001, NP = 100, itermax = 100),
  seed = NULL,
  free = NULL,
  n_restarts = 1
)
```

Arguments

`model`	An object of class `xslMod` giving the model to cross-validate (its starting parameter values are only used to determine how many parameters `xsl_fit()` should optimize).
`combined_data`	A list of `xslData` conditions used for cross-validation.
`lower`	Numeric vector of lower bounds for the model's parameters.
`upper`	Numeric vector of upper bounds for the model's parameters.
`n_folds`	Number of folds (default: 5).
`control`	Control arguments passed to `xsl_fit()`/`xsl_run()`.
`deoptim_control`	Control parameters passed to `DEoptim()` (via `xsl_fit()`).
`seed`	Optional random seed, for reproducible fold assignment and fits.
`free`, `n_restarts`	Passed to `xsl_fit()`: which parameters the bounds are for, and how many DEoptim runs to keep the best of, per fold.

Value

A list containing cross-validation results:

- `folds`: a list (one per fold) with the fitted `params`, `train_sse` (in-sample), and `test_sse` (out-of-sample).
- `params`: a matrix of fitted parameters, one row per fold.
- `train_sse` / `test_sse`: numeric vectors of per-fold SSE.
- With a likelihood objective (`control = xslControl(objective = "binomial")` or `"multinomial"`), also `train_nll` / `test_nll`: per-fold negative log-likelihoods, in-sample (what was minimized) and out-of-sample. Summed over folds, `test_nll` scores every condition once, at parameters fit without it.

Examples

```
# A reduced DEoptim search keeps this example fast; drop
# deoptim_control for a real, thorough fit.
cross_validated_group_fits(decay(C = 0.98), xsl_datasets[1:6],
                           lower = 0.8, upper = 1.0, n_folds = 2,
                         deoptim_control = DEoptim::DEoptim.control(NP = 10, itermax = 5))
```

`decay` *Decay model*

Description

Decay model

Usage

```
decay(C)
```

Arguments

`C`	decay

Value

An object of class xslMod

Examples

```
mod <- decay(C = 0.5)
xsl_run(mod, get_example_ambiguous_condition())
```

fazly *Fazly et al. 2010 probabilistic associative model*

Description

Fazly et al. 2010 probabilistic associative model

Usage

```
fazly(lambda, beta, alpha = 10, epsilon = 0.001, theta = NULL)
```

Arguments

lambda	Small smoothing factor (e.g. 1e-5)
beta	Upper bound on number of symbol types to expect (e.g. 8500)
alpha	Fixed smoothing parameter (e.g. 10)
epsilon	Fixed smoothing parameter (e.g. 0.001)
theta	Threshold to determine when associations enter the lexicon (defaults to NULL, i.e. model variant without thresholding).

Value

An object of class xslMod

Examples

```
mod <- fazly(lambda = 1e-5, beta = 8500)
xsl_run(mod, get_example_ambiguous_condition())

mod_t <- fazly(lambda = 1e-5, beta = 8500, theta = 0.7)
xsl_run(mod_t, get_example_ambiguous_condition())
```

fgt2009 *Frank, Goodman & Tenenbaum (2009) intentional Bayesian word-learning model*

Description

A joint Bayesian model that infers a *lexicon* – a set of word-object edges – from the whole corpus at once, marginalizing over the speaker's latent referential intention on each trial. Words are generated either referentially (probability gamma, naming an object in the intention) or non-referentially (1 - gamma, with in-lexicon words down-weighted by kappa); a geometric prior P(L) proportional to exp(-alpha |L|) favors small lexicons. Inference is MCMC (blocked per-object Gibbs plus MI-biased single-edge Metropolis moves), and xslFit$matrix holds the resulting posterior edge marginals P(word names object | corpus).

Usage

```
fgt2009(
  alpha,
  gamma = 1,
  kappa = 0.5,
  n_chains = 4L,
  n_warmup = 120L,
  n_samples = 300L,
  gibbs_per = 1L,
  edge_per = 1L,
  top_k = 6L,
  seed = 1L
)
```

Arguments

alpha	Geometric lexicon-size prior: P(L) proportional to exp(-alpha |L|). Must be tuned to corpus size.
gamma	Probability that a word is used referentially. 1 (the default) is appropriate when every word heard is a label, as in controlled XSL experiments.
kappa	Down-weight applied to an in-lexicon word used non-referentially (only relevant when gamma < 1).
n_chains, n_warmup, n_samples, gibbs_per, edge_per, top_k, seed	Sampler controls. Defaults are tuned for the small, dense corpora of XSL experiments; raise n_chains/n_samples to reduce Monte Carlo noise.

Details

This is a **batch** model, unlike the incremental associative models in this package: it has no trial-by-trial learning trajectory, and control$reps is ignored. Ported from the Python package wordlearn (github.com/mcfrank/wurwur); see also fgt2009_rsa() for the pragmatic extension.

alpha scales with corpus size (larger corpora want a larger alpha) and the fit-vs-alpha curve is single-peaked, so alpha is best chosen with a sweep (see fgt2009_sweep_alpha()) rather than a general-purpose optimizer. gamma and kappa default to the values used for the analogous cross-situational simulation in the source package (gamma = 1: every word in a controlled XSL experiment is a referential label, so there is no non-referential channel; kappa = 0.5). The naturalistic-speech values are gamma = 0.1, kappa = 0.05.

Value

An object of class xslMod

Examples

```
mod <- fgt2009(alpha = 1)
xsl_run(mod, get_example_ambiguous_condition())
```

`fgt2009_rsa` *Frank, Goodman & Tenenbaum word-learning model with pragmatic (RSA) reasoning*

Description

`fgt2009()` with the literal speaker replaced by a Rational Speech Act pragmatic speaker: a speaker who chooses words to be informative to a listener who reasons about that speaker (Frank & Goodman, 2012; Smith, Goodman & Frank, 2013). The non-referential channel, the `kappa` penalty, and the marginalization over intentions are unchanged, so the literal `fgt2009()` model is the `rsa_depth = 0` special case. The recursion is what produces *strong* mutual exclusivity – the speaker could have used a different word – which the literal model underproduces.

Usage

```
fgt2009_rsa(
  alpha,
  gamma = 1,
  kappa = 0.5,
  rsa_alpha = 3,
  rsa_depth = 1L,
  n_chains = 4L,
  n_warmup = 120L,
  n_samples = 300L,
  gibbs_per = 1L,
  edge_per = 1L,
  top_k = 6L,
  seed = 1L
)
```

Arguments

`alpha`	Geometric lexicon-size prior: `P(L)` proportional to `exp(-alpha |L|)`. Must be tuned to corpus size.
`gamma`	Probability that a word is used referentially. `1` (the default) is appropriate when every word heard is a label, as in controlled XSL experiments.
`kappa`	Down-weight applied to an in-lexicon word used non-referentially (only relevant when `gamma < 1`).
`rsa_alpha`	Speaker rationality (softmax temperature) in the RSA recursion.
`rsa_depth`	Number of pragmatic recursions; 1 gives the usual one-step pragmatic listener/speaker.
`n_chains`, `n_warmup`, `n_samples`, `gibbs_per`, `edge_per`, `top_k`, `seed`	Sampler controls. Defaults are tuned for the small, dense corpora of XSL experiments; raise `n_chains`/`n_samples` to reduce Monte Carlo noise.

Value

An object of class xslMod

Examples

```
mod <- fgt2009_rsa(alpha = 1, rsa_alpha = 3)
xsl_run(mod, get_example_ambiguous_condition())
```

`fgt2009_sweep_alpha` *Sweep the lexicon-size prior alpha for an FGT model*

Description

`fgt2009()`'s `alpha` trades lexicon size against fit and must scale with corpus size, and the fit-vs-`alpha` curve is single-peaked – so a coarse sweep is the right way to choose it (and to evaluate the model fairly), rather than a general-purpose optimizer. This runs `xsl_run()` for each `alpha` and returns the per-value SSE against the dataset's human accuracy.

Usage

```
fgt2009_sweep_alpha(
  data,
  alphas,
  rsa = FALSE,
  gold = NULL,
  ...,
  control = xslControl()
)
```

Arguments

`data`	An `xslData` object (or list of them).
`alphas`	Numeric vector of `alpha` values to try.
`rsa`	Logical; if `TRUE`, sweep `fgt2009_rsa()` instead of `fgt2009()`.
`gold`	Optional gold-standard lexicon as `list(words, objects)`. When given, the result gains an `f_max` column (best-threshold F-score against `gold`). `data` must be a single `xslData` in this case.
`...`	Further arguments to the model constructor (e.g. `gamma`, `kappa`, sampler controls).
`control`	Control arguments passed to `xsl_run()`.

Details

For a naturalistic corpus (e.g. rollins_corpus, fm_corpus) there is no human accuracy vector; pass `gold` (a `list(words, objects)` lexicon) to also get the best-threshold F-score of the learned matrix against it (via `get_roc_max()`) at each `alpha`. Remember such corpora also want the naturalistic-speech `gamma = 0.1`, `kappa = 0.05` rather than the defaults (see `fgt2009()`).

Value

A data frame with columns `alpha` and `sse` (plus `unweighted_sse` when `data` is a list, or `f_max` when `gold` is given).

Examples

```
fgt2009_sweep_alpha(get_example_ambiguous_condition(),
                    alphas = c(1, 2, 4, 8))
```

`fm_corpus` *Frank, Tenenbaum & Fernald naturalistic word-learning corpus*

Description

4763 caregiver utterances (24 mother-infant sessions), each paired with the objects present in the scene. 1122 word types, 30 object types present in scenes; roughly half the utterances are non-referential. Unlike rollins_corpus, the speaker's referential intention is hand-coded per utterance.

Usage

```
fm_corpus
```

Format

An object of class `list` of length 5.

Details

A list with:

`data` an xslData object with the 4763 training utterances (character-vector `words`/`objects` per trial). No `accuracy`/`test`.

`intents` a length-4763 list; `intents[[t]]` is the object(s) the speaker referred to on utterance `t` (character vector, empty for non-referential utterances). A per-utterance gold signal.

`gold` a hand-curated gold lexicon as `list(words, objects)` (41 pairs), for `get_fscore()` / `get_roc()` / `get_tp()`.

`gold_variants` `list(strict, permissive)` – two auto-derived alternatives (39 and 116 pairs) from looser vs. tighter co-occurrence thresholds on the coded intentions.

`reference` the citation string.

Not part of xsl_datasets. Gaze/hand attentional cues present in the source CSVs are not imported. Imported from the wurwur package (`https://github.com/mcfrank/wurwur`); see `data-raw/add_wurwur_corpora.R`.

Source

Frank, M. C., Tenenbaum, J. B., & Fernald, A. (2013). Social and discourse contributions to the determination of reference in cross-situational word learning. *Language Learning and Development, 9*(1), 1-24.

Examples

```
length(fm_corpus$data$train$words)
fm_corpus$intents[[1]]
fm_corpus$gold_variants$strict$words
```

`gangwani2011_category` *Gangwani, Kachergis & Yu simultaneous category + object name learning*

Description

Experiment 1 of Gangwani, Kachergis & Yu, "Simultaneous Cross-situational Learning of Category and Object Names" (an undergraduate-journal version is Gangwani & Kachergis, *Indiana Undergraduate Journal of Cognitive Science* 5, 2010). A **hierarchical** cross-situational design: on every training trial the learner sees 2 objects and hears 3 words – the two objects' own 1-to-1 names *plus* one 1-to-many "category" label that names those two objects and two further objects seen on other trials. Every object is thus part of a 2-to-1 word-referent mapping (a basic name and a category label), and the paper's finding is that learners violate mutual exclusivity: they learn *both* labels for the same object.

Usage

```
gangwani2011_category
```

Format

An object of class `list` of length 3.

Details

A named list of three xslData objects, one per training block – `"block 1 (ad hoc)"`, `"block 2 (natural)"`, `"block 3 (ad hoc)"` – run in that fixed order. All three share the same 36-trial co-occurrence schedule (only the stimuli differ); each carries its own block-specific human data.

Per object: `train$words[[t]]` is `c(name_i, name_j, category_label)` and `train$objects[[t]]` is `c(i, j)`. Words 1-12 are object names (name `i` pairs with object `i`); words 13-15 are the category labels (13/14/15 name objects 1-4 / 5-8 / 9-12). `accuracy` is per-word human P(correct referent | word), length 15: positions 1-12 are 12-AFC name accuracy, positions 13-15 are 3-AFC category-membership accuracy (n = 34 canonical-block-order participants; the paper reports 33, dropping one more low performer). `response_matrix` (15 x 12, row-normalised) is the full human choice distribution including the category labels. `test` holds only the 12 name trials (12-AFC).

Not part of xsl_datasets, and not scorable by the built-in `xsl_run()` SSE: the association matrix is 15 words x 12 objects, so `get_perf()` / `mafc_test()` (which score via the diagonal `m[w, w]`) cover only words 1-12. Score the category labels – and the ME-violation measure that is the paper's point – with `score_gangwani2011_category()` in `tests/bakeoff_comparison/gangwani2011_category_fit.R`. The natural-category feature-similarity effect (block 2) and generalization to novel objects are out of scope for the package's amodal, index-based models and are not represented. See `data-raw/add_gangwani2011_category.` for construction details.

Examples

```
block1 <- gangwani2011_category[["block 1 (ad hoc)"]]
# 15 x 12 matrix; suppressWarnings() because get_perf()'s diagonal rule
# warns on the non-square matrix (the built-in SSE is not meaningful here).
m <- suppressWarnings(xsl_run(baseline(), block1))$fits[[1]]$matrix
mafc_test(m, block1$test)                          # names (words 1-12) only
```

`get_crossvalidated_model_fit` *Get cross-validated model fit*

Description

Performs k-fold cross-validation on a model using the available datasets: the datasets are split into `n_folds` groups, and for each fold the model is fit (via `xsl_fit()`) to the other folds and evaluated (via `xsl_run()`) on the held-out fold.

Usage

```
get_crossvalidated_model_fit(
  model_name,
  n_folds = 5,
  datasets = NULL,
  control = xslControl(),
 deoptim_control = DEoptim::DEoptim.control(reltol = 0.001, NP = 100, itermax = 100),
  n_restarts = NULL,
  seed = NULL,
  fit_lapse = FALSE
)
```

Arguments

`model_name`	Name of the model to cross-validate (see `show_models()`)
`n_folds`	Number of folds for cross-validation (default: 5)
`datasets`	Optional list of datasets to use (defaults to all available)
`control`	Control arguments passed to `xsl_run()`/`xsl_fit()`; notably `n_sim`, the number of simulations averaged per evaluation of a stochastic model (default 500 – lowering this can speed up cross-validation of stochastic models substantially).
`deoptim_control`	Control parameters passed to `DEoptim()` (via `cross_validated_group_fits()`).
`n_restarts`	Number of DEoptim runs to keep the best of, per fold; defaults to the model's registry default (see `get_group_model_fit()`).
`seed`	Optional random seed for reproducible folds and fits.
`fit_lapse`	With a likelihood objective (`control = xslControl(objective = "binomial")` or `"multinomial"`), also fit the lapse rate, in [0, 1].

Value

A list containing cross-validation results (see `cross_validated_group_fits()`)

Examples

```
# Get cross-validated fit for the decay model. A reduced DEoptim search
# keeps this example fast; drop deoptim_control for a real, thorough fit.
cv_fit <- get_crossvalidated_model_fit(
  "decay", n_folds = 2, datasets = xsl_datasets[1:6],
  deoptim_control = DEoptim::DEoptim.control(NP = 10, itermax = 5))
```

---

get_example_ambiguous_condition

Get example ambiguous condition

Description

This function creates an example condition with two trials, and two word-object pairs per trial. Each word-object pair appears only once, and thus each word has a 50% chance of being associated with the correct referent. The function returns a list of the trials with nested words and objects per trial, as well as a vector of the conditional probability of correctly selecting the correct referent given each word.

Usage

```
get_example_ambiguous_condition()
```

Value

An object of class `xslData` containing a list of training trials (with nested words and objects per trial) and accuracy (P(correct referent | word)).

Examples

```
get_example_ambiguous_condition()
```

get_example_unambiguous_condition

Get example unambiguous condition

Description

This function creates an example condition with three trials and two word-object pairs per trial. Each word-object pair appears twice, and so if the most frequently co-occurring referent for each word is selected then all three words would be learned.

Usage

```
get_example_unambiguous_condition()
```

Value

An object of class `xslData` containing a list of training trials (with nested words and objects per trial) and accuracy (P(correct referent | word)).

Examples

```
get_example_unambiguous_condition()
```

get_fscore

Calculate F-score, precision, recall, and specificity for a knowledge matrix at a given threshold

Description

This function calculates the F-score, precision, recall, and specificity for a given knowledge matrix at a specified threshold. It uses the concept of true positives, false positives, and false negatives, determined from the knowledge matrix and an optional gold lexicon. The function is useful for evaluating the performance of a model in terms of its ability to correctly identify associations between words and referents.

Usage

```
get_fscore(m, threshold, gold_lexicon = NULL)
```

Arguments

`m`	A matrix representing the knowledge matrix with words as rows and referents as columns.
`threshold`	A numeric value representing the threshold for considering an association between a word and a referent as positive.
`gold_lexicon`	Optional; a data frame or list where each row/element represents a word-object pair in the gold lexicon. If provided, it is used to calculate true positives, false positives, and false negatives.

Value

A tibble with columns for threshold, precision, recall, specificity, and F-score.

Examples

```
dat <- xsl_datasets[[1]]
x <- xsl_run(baseline(), dat)
mat <- x$fits[[1]]$matrix
lex <- list(words = rep(1:18), objects = rep(1:18))
get_fscore(mat, 0.5, lex)
```

---

get_group_model_fit *Get group model fit for a specific model*

Description

Fits a model to all available datasets and returns the group-level fit. This function provides a convenient way to get pre-computed model fits for common models.

Usage

```
get_group_model_fit(
  model_name,
  datasets = NULL,
  control = xslControl(),
 deoptim_control = DEoptim::DEoptim.control(reltol = 0.001, NP = 100, itermax = 100),
  n_restarts = NULL,
  seed = NULL,
  fit_lapse = FALSE
)
```

Arguments

model_name	Name of the model to fit (see show_models())
datasets	Optional list of datasets to fit to (defaults to all available)
control	Control arguments passed to xsl_run() (via xsl_fit()); notably n_sim, the number of simulations averaged per evaluation of a stochastic model (default 500 – lowering this can speed up fitting of stochastic models substantially).
deoptim_control	Control parameters passed to DEoptim() (via xsl_fit()).
n_restarts	Number of DEoptim runs to keep the best of; defaults to the model's registry default (5 for uncfam_predictive, 3 for tilles, otherwise 1).
seed	Optional random seed for reproducible fits.
fit_lapse	With a likelihood objective (control = xslControl(objective = "binomial") or "multinomial"), also fit the lapse rate, in [0, 1].

Value

A list containing the fitted model results

Examples

```
# Get group fit for the uncfam model. A small dataset subset and a
# reduced DEoptim search keep this example fast; drop those arguments
# for a real, thorough fit.
group_fit <- get_group_model_fit(
  "uncfam", datasets = xsl_datasets[1:3],
  deoptim_control = DEoptim::DEoptim.control(NP = 10, itermax = 5))
```

---

get_perf *Calculate Luce choice (proportion correct) for each item in a model knowledge matrix*

Description

This function computes the Luce choice, or the proportion of correct selections, for each item in a given model knowledge matrix. It assesses the probability of correctly identifying each referent based on the knowledge matrix, providing a measure of model performance per item.

Usage

```
get_perf(m, d = NULL)
```

Arguments

m	A square matrix representing the model's knowledge, where rows correspond to words and columns correspond to referents. The diagonal elements represent correct associations, and off-diagonal elements represent incorrect associations.
d	Exponent for exponentiated choice rule

Details

Negative associations (possible in error-driven models such as `rescorla_wagner()` and `kalman_filter()`) are treated as 0 before the choice rule is applied – Luce choice is undefined for negative values, and would otherwise yield "probabilities" outside [0, 1] – matching `predict_referent()`. A word whose associations are then all 0 is answered at chance (`1 / ncol(m)`).

Value

A named numeric vector where each element corresponds to an item in the matrix. The value of each element represents the proportion of correct selections (Luce choice) for that item, calculated as the ratio of the correct association (diagonal element) to the total associations for that item.

Examples

```
x <- xsl_run(baseline(), get_example_ambiguous_condition())
mat <- x$fits[[1]]$matrix
get_perf(mat)
```

`get_roc` *Calculate receiver operating characteristic (ROC) scores for a model association Matrix*

Description

This function computes receiver operating characteristic (ROC) scores for a given model association matrix. It evaluates the performance of the model at various thresholds, providing metrics like f-scores, precision, and recall. The function can operate with a range of thresholds and optionally consider a gold lexicon for calculating true and false positives and negatives. The result is a comprehensive assessment of model performance over a continuum of classification thresholds.

This function computes the maximum F-score from the receiver operating characteristic (ROC) scores of a model. It leverages the `get_roc()` function to calculate the ROC scores and then extracts the highest F-score, providing a concise metric for the best classification performance of the model.

Usage

```
get_roc(m, thresholds = seq(0, 1, 0.01), gold_lexicon = NULL)

plot_roc(m, thresholds = seq(0, 1, 0.01), gold_lexicon = NULL)

get_roc_max(m, thresholds = seq(0, 1, 0.01), gold_lexicon = NULL)
```

Arguments

`m`	A matrix representing the knowledge matrix with words as rows and referents as columns.
`thresholds`	Vector of thresholds to use.
`gold_lexicon`	Optional; a data frame or list where each row/element represents a word-object pair in the gold lexicon. If provided, it is used to calculate true positives, false positives, and false negatives.

Value

A tibble with columns for the threshold, f-score, precision, and recall.

A single numeric value representing the maximum F-score obtained from the ROC scores of the model.

Examples

```
dat <- xsl_datasets[[1]]
x <- xsl_run(baseline(), dat)
mat <- x$fits[[1]]$matrix
lex <- list(words = rep(1:18), objects = rep(1:18))
get_roc(mat, gold_lexicon = lex)
plot_roc(mat, gold_lexicon = lex)
get_roc_max(mat, gold_lexicon = lex)
```

`get_tp` *Get true positives (TP), given a knowledge matrix and a gold-standard lexicon*

Description

This function iterates over words in a given gold lexicon and accumulates the associative strength (can be integral e.g. 1, or real-valued) in the knowledge matrix for the intended referents (present in the gold lexicon). Returns the number of expected true positives (TP) for this gold lexicon and knowledge matrix.

Usage

```
get_tp(m, gold_lexicon)
```

Arguments

`m`	A matrix representing the knowledge matrix with words as rows and referents as columns.
`gold_lexicon`	A data frame or list where each row/element represents a word-object pair in the gold lexicon.

Value

A single value with the expected number of true positives.

Examples

```
dat <- xsl_datasets[[10]]
x <- xsl_run(baseline(), dat)
mat <- x$fits[[1]]$matrix
lex <- list(words = rep(1:18), objects = rep(1:18))
get_tp(mat, lex)
```

get_train_test_split *Split Data into Training and Testing Sets Based on Indices*

Description

This function divides a given set of conditions into training and testing sets. The division is based on a provided vector of indices, which specify the conditions to be used for testing.

Usage

```
get_train_test_split(test_inds, conds)
```

Arguments

test_inds	A numeric vector of indices indicating which elements in conds should be used for the test set.
conds	A list of conditions (e.g. xslData objects), where each condition is represented as a separate element.

Value

A list with two elements: train and test. Each element is a list of conditions, where train includes the conditions not indexed by test_inds, and test includes the conditions indexed by test_inds.

Examples

```
split <- get_train_test_split(c(1, 3), xsl_datasets[1:5])
length(split$train)
length(split$test)
```

guess_and_test *Guess and test model*

Description

Trueswell & Gleitman 2011 guess-and-test model

Usage

```
guess_and_test(f, sa)
```

Arguments

f	forget at retrieval
sa	prob of storage (slow learning down)

Value

An object of class xslMod

Examples

```
mod <- guess_and_test(f = .1, sa = .5)
xsl_run(mod, get_example_ambiguous_condition())
```

`kachergis2012_highlighting`

Kachergis (2012) highlighting experiment data

Description

A list of two `xslData` objects ("words as cues" and "objects as cues") from the highlighting experiment in Kachergis (2012, CogSci), "Learning Nouns with Domain-General Associative Learning Mechanisms". Training trials (genuine simultaneous 2-cue trials, matching the paper's Figure 2: 2 words + 1 object, or 2 objects + 1 word, per trial) are read directly from the real per-trial ordering file for the real N=67 dataset behind the paper's published statistics – confirmed by reproducing its exact reported response proportions and chi-square values. See `data-raw/add_kachergis2012_highlighting.R` for the full construction rationale, including index relabeling needed to work around a `uncfam_model()` indexing limitation.

Usage

```
kachergis2012_highlighting
```

Format

An object of class `list` of length 2.

Details

Each condition replicates the classic 3-role highlighting structure (PE, PL, I) twice. Words-as-cues' PL items have their real target at a different index than their own word index, and its I items have two legitimate targets – neither fits `xslData`'s one-correct-object-per- word `accuracy` convention, so only its two PE items get a real accuracy value (the rest are `NA`). Objects-as-cues has no such issue (every tested word's own index is among its trained objects after relabeling), so all 4 of its items have real accuracy. Because of the partial-NA accuracy vector, this dataset is kept standalone rather than appended to xsl_datasets: an `NA` would silently poison every other dataset's aggregate fit in `get_group_model_fit()`/`get_crossvalidated_model_fit()`, which sum SSE across all of `xsl_datasets`. See `tests/bakeoff_comparison/kachergis2012_highlighting_fit.R` for a custom scorer that fits a model against all four of the paper's reported proportions (PE-E, PL-L, I-E, I-L) directly off the association matrix, the way the paper's own (non-package) fitting procedure did.

kachergis_initial_accuracy

Kachergis, Grimmick, & Gureckis initial accuracy experiment data

Description

A list of two `xslData` objects ("High Initial Accuracy" and "Low Initial Accuracy") from an unpublished MTurk cross-situational word learning experiment manipulating how many of 18 words are first shown ("familiarized") with a different object than the one they are then cross-situationally studied with. The study pairing is the test-correct one, and is on the diagonal (`m[w, w]`); an initially inaccurate word's familiarization object is off-diagonal. The first 18 training trials are the familiarization phase. Not included in xsl_datasets (see `data-raw/add_kachergis_initial_accuracy.R` for construction details and `tests/bakeoff_comparison/kachergis_initial_accuracy_fit.R` for an example model comparison).

Usage

```
kachergis_initial_accuracy
```

Format

An object of class `list` of length 2.

kachergis_priming

Kachergis, Yu & Shiffrin priming experiments: 4-block learning trajectories

Description

Three experiments run alongside Kachergis, Yu & Shiffrin's frequency and contextual-diversity studies. Participants studied the same cross-situational training sequence four times, taking an 18AFC test on all 18 words after each repetition. Each condition is therefore a multi-block xslData: `train` is one block, `accuracy` is a 4 x 18 matrix (one row per block), and `response_matrix` is a list of four 18 x 18 confusion matrices. `xsl_run()` scores block `b` on a model trained on `b` repetitions of the sequence, so a model's learning trajectory can be compared with people's.

Usage

```
kachergis_priming
```

Format

An object of class `list` of length 3.

Details

A named list of three conditions:

- `"4x4 6x"`: 4 words and 4 objects per trial, 27 trials, every pair 6 times (n = 54). The sequence is that of the original 4x4 condition, which differs from `xsl_datasets` condition `223` only in words 4 and 16 trading places between trials 11 and 24.
- `"4x4 3/6/9x"`: 4 words and 4 objects per trial, 27 trials, 6 pairs each 3, 6 and 9 times (n = 44); the training of `3_x8_369_4x4`.
- `"3x3 3/6/9x"`: 3 words and 3 objects per trial, 36 trials, 6 pairs each 3, 6 and 9 times with low contextual diversity (n = 66); the training of `freq369-3x3loCD`.

The participants are different from those in `xsl_datasets` (until XSLmodels PR #30, their first block had been pooled into `3_x8_369_4x4` and `freq369-3x3loCD`). Participants whose accuracy across all four blocks was below 2/18 were excluded in the original analysis. Not included in xsl_datasets; see `data-raw/add_kachergis_priming.R` for construction details.

Examples

```
d <- kachergis_priming[["4x4 6x"]]
d
rowMeans(d$accuracy)   # people's learning curve
fit <- xsl_run(uncfam(X = .2, B = 3, C = .97), d)
rowMeans(fit$fits[[1]]$perf)   # the model's
```

---

kalman_filter *Kalman filter associative model*

Description

A Kalman-filter generalization of `rescorla_wagner()`'s error-driven update rule (Dayan & Kakade, 2001; Kruschke, 2008; Gershman, 2015): word- object associations are tracked as Gaussian beliefs, and each trial's update is scaled by a Kalman gain, `sigma2 / (sigma2 + sigma2_obs)`. This makes the effective learning rate adaptive rather than fixed – large (fast learning) while an association is still uncertain, and automatically shrinking as confidence accumulates – and it emerges from three interpretable parameters rather than being hand-set. Between observations, every association's variance grows by `tau2` (the model assumes true associations drift slowly over time), which keeps the model able to revise a belief rather than converging to a fixed point.

Usage

```
kalman_filter(
  tau2,
  sigma2_obs,
  sigma2_0,
  covariance = c("independent", "full"),
  outcomes = c("present", "all"),
  phi = NULL
)
```

Arguments

`tau2`	Process (diffusion) noise: how much uncertainty about every association grows per trial, whether or not it was observed
`sigma2_obs`	Observation noise: how uninformative a single co-occurrence observation is (larger = slower learning per trial)
`sigma2_0`	Initial (prior) uncertainty about every association, before any training
`covariance`	`"independent"` (the default): one filter per word-object pair. `"full"`: one filter per object, over all words, with a full covariance.
`outcomes`	`"present"` (the default) updates only the objects present on a trial; `"all"` also updates absent objects toward 0.
`phi`	Optional softmax inverse temperature for the choice rule. `NULL` (the default) uses Luce choice.

Details

By default (`covariance = "independent"`), every word-object association is its own one-dimensional filter that observes 1 whenever the pair co-occurs. Associations then don't interact: there is no competition between a trial's words, and an association is a saturating function of how often and how recently its pair co-occurred. With a small gain it is simply co-occurrence counting.

Value

An object of class xslMod

Options

- `covariance = "full"` is the Kalman filter of the associative-learning literature (Dayan & Kakade, 2001; Kruschke, 2008). Each object (outcome) has a weight vector over words with a full covariance matrix. The present words jointly predict the object (the sum of their weights), so they share one prediction error, and the gain is a vector over words. Words that appear together come to have negatively correlated weights, so later evidence about one revises the other: explaining away and backward blocking (retrospective revaluation), as when learning that one word on an earlier trial named an object implies the trial's other words did not. Memory and time grow with the square of the vocabulary.
- `outcomes = "all"` treats every object as an outcome on every trial, observing 1 if it is present and 0 if absent, as in the canonical filter, rather than updating only the objects present. Associations can then become negative, which Luce choice treats as 0; `phi` gives a choice rule defined for them.
- `phi` replaces Luce choice with a softmax choice rule with inverse temperature `phi` (see `softmax_weights()`).

Note that the three noise parameters share a scale redundancy: multiplying `tau2`, `sigma2_obs` and `sigma2_0` together by any constant leaves every Kalman gain – and hence the learned matrix – unchanged. Only two of the three are identifiable; when fitting, hold one fixed (conventionally `sigma2_obs`).

References

Dayan, P., & Kakade, S. (2001). Explaining away in weight space. In *Advances in Neural Information Processing Systems 13* (pp. 451-457).

Gershman, S. J. (2015). A unifying probabilistic view of associative learning. *PLoS Computational Biology, 11*(11), e1004567.

Kruschke, J. K. (2008). Bayesian approaches to associative learning: From passive to active learning. *Learning & Behavior, 36*(3), 210-226.

Examples

```
mod <- kalman_filter(tau2 = .01, sigma2_obs = .5, sigma2_0 = 1)
xsl_run(mod, get_example_ambiguous_condition())
# the full-covariance filter, with absent objects as outcomes
mod <- kalman_filter(tau2 = .001, sigma2_obs = 1, sigma2_0 = .5,
                     covariance = "full", outcomes = "all")
xsl_run(mod, get_example_ambiguous_condition())
```

---

koehne2013 *Koehne, Trueswell, & Gleitman (2013) multiple-meaning word learning*

Description

Four conditions from Koehne, Trueswell, & Gleitman (2013), *Multiple proposal memory in observational word learning*, Proceedings of the 35th Annual Conference of the Cognitive Science Society: 8 words, 1 word and 4 objects per trial, each word with one referent present whenever it is heard and one present half the time; the order of trials with (P) and without (A) the 50% referent is blocked (AAAPPP, PPPAAA) or alternating (APAPAP, PAPAPA).

Usage

```
koehne2013
```

Format

A named list of 4 xslData objects, one per condition.

Details

Only each condition's reported mean accuracy is available (no per-item or per-participant data), so `accuracy` is that mean replicated across items. Held out of xsl_datasets (see xsl_holdout_datasets), as in the original bake-off: use it to test whether a model fit elsewhere generalizes, by comparing its mean predicted accuracy with the reported mean.

Examples

```
sapply(koehne2013, \(d) d$accuracy[1])
```

mafc_test *Evaluate m-alternative forced choice test*

Description

Scores a set of test trials against a model's word-object matrix, returning the probability of choosing the correct object on each trial. Each trial presents one word and a set of candidate referents; the correct object is the one whose id matches the word's id (the package's diagonal convention), unless the test gives an explicit `correct` object for that trial (e.g. to test a word's association with a second, co-occurring referent, with its own referent absent). A thin wrapper over `predict_referent()`.

Usage

```
mafc_test(m, test, ...)
```

Arguments

`m`	A matrix representing model memory with words as rows and referents as columns.
`test`	A list representing the test trials: `words` (one word per trial), `objects` (the candidate referents on each trial), and optionally `correct` (the correct referent on each trial; defaults to the word's own id).
`...`	Further arguments passed to `predict_referent()` (e.g. `pragmatic`, `threshold`, or `d`, the exponent of a power choice rule).

Value

A vector with the probability of choosing the correct object, given each word.

Examples

```
dat <- xsl_datasets[[10]]
x <- xsl_run(baseline(), dat)
mat <- x$fits[[1]]$matrix
mafc_test(mat, dat$test)
```

minerva2 *MINERVA2 episodic memory model*

Description

Hintzman's (1984, 1986) MINERVA2 adapted to cross-situational word learning: each trial is stored as a single episodic trace formed by concatenating that trial's word-vector sum and object-vector sum (words and objects are random Gaussian vectors, one per vocabulary item). At test, a word probe (word-half only, object-half zeroed) activates every trace by its similarity cubed; the activation-weighted sum of traces (the "echo") is compared, on its object half, against every candidate object to give a graded association – exponentiated by `X` into a Luce-choice-ready matrix rather than MINERVA2's usual hard best-match.

Usage

```
minerva2(X, D = 100)
```

Arguments

X	Choice-rule temperature: how sharply the echo/object cosine similarity is exponentiated into a choice weight (larger = more winner-take-all)
D	Dimensionality of the random word/object feature vectors (fixed hyperparameter, not a free cognitive parameter)

Details

Performance note: memory is exact (no forgetting/decay), so the model keeps one `2*D`-length trace per trial and a test pass costs `O(voc_sz * n_traces * D)`. `D` is a fixed representational hyperparameter (not usually treated as a free cognitive parameter), not included by default in this package's DEoptim registry bounds – see `xsl_model_registry()`.

Value

An object of class xslMod

Examples

```
mod <- minerva2(X = 5, D = 50)
xsl_run(mod, get_example_ambiguous_condition())
```

---

multi_sampling *Multi-sampling associative model*

Description

Multi-sampling associative model

Usage

```
multi_sampling(C, X, B, K = 1)
```

Arguments

C	Decay
X	Total associative weight to distribute per trial
B	Associative weight to distribute each sample
K	Number of associations to update per word (constraint: B <= X)

Value

An object of class xslMod

Examples

```
mod <- multi_sampling(C = 1, X = .2, B = .3, K = 1)
xsl_run(mod, get_example_ambiguous_condition())
```

`new_xslFit` *Constructor for xsl_model S3 class*

Description

Constructor for xsl_model S3 class

xslMod S3 class

Constructor for xslMod S3 class

Usage

```
new_xslFit(x = list())

xslMod(
  name = character(),
  description = character(),
  model,
  params = numeric(),
  stochastic = logical(),
  supports_start_matrix = FALSE
)

new_xslMod(x = list())
```

Arguments

`x`	List with elements perf, matrix, traj, sse
`name`	Name
`description`	Description
`model`	Model fitting function
`params`	List of parameters
`stochastic`	Logical indicating whether model is stochastic
`supports_start_matrix`	Logical: does `model` initialize from `control$start_matrix`? `xsl_run()` errors if a start matrix is supplied to a model that doesn't, rather than letting it be silently ignored.

Value

An object of class xslMod

plot_training_trials *Create Animated Plot of Co-Occurrence Matrix from Training Trials*

Description

This function takes a sequence of training trials, each consisting of words and objects, and creates an animated plot of the word-object co-occurrence matrix. The plot evolves over each trial, visually representing the accumulation of co-occurrences. The animation is saved to a file if a filename is provided.

Usage

```
plot_training_trials(train, filename = NULL)
```

Arguments

`train`	A list representing the training data, where each element is a trial containing lists of words (`words`) and objects (`objects`).
`filename`	A string specifying the filename where the animation should be saved. If NULL, no file will be saved.

Value

A long-format data frame where each row corresponds to a trial and includes columns for trial number, word, object, and the count of co-occurrences up to that trial.

Examples

```
plot_training_trials(xsl_datasets[[1]]$train)
```

predict_referent *Predict referent selection for a single trial from an association matrix*

Description

Given a model's word-object association matrix, returns the probability distribution over a trial's candidate objects for one heard word – the quantity to compare against a participant's choice on a referent-selection (m-alternative forced choice) trial. This is model-agnostic: it works on the `matrix` of any `xslFit` (or any word-by-object matrix), reading each entry as an association strength / meaning weight.

Usage

```
predict_referent(
  m,
  word,
  objects,
  prior = NULL,
  pragmatic = FALSE,
  threshold = NULL,
  rsa_alpha = 3,
  depth = 1,
  d = NULL
)
```

Arguments

`m`	A word-by-object association matrix (e.g. `xsl_run(mod, data)$fits[[1]]$matrix`). Assumed non-negative; negative entries are clamped to 0. `dimnames` are used to resolve character `word`/`objects`.
`word`	The heard word: a row name of `m`, or a positive integer row index. A value not present in `m` is treated as a novel word.
`objects`	The candidate objects present on the trial: column names of `m`, or positive integer column indices. An entry not present in `m` is treated as an unnamed (novel) object.
`prior`	Optional prior over `objects` (length `length(objects)`, non-negative). Defaults to uniform. Set an entry to 0 to remove that object from consideration entirely (e.g. an object the speaker has no epistemic access to).
`pragmatic`	If `TRUE`, use the RSA pragmatic listener instead of the literal one.
`threshold`	Optional; binarize `m` at this value (`m >= threshold`) before predicting. The RSA layer is sharpest on a binary lexicon.
`rsa_alpha, depth`	Speaker rationality and recursion depth for the RSA listener (only used when `pragmatic = TRUE`).
`d`	Optional exponent for an exponentiated (power) choice rule: the (clamped, possibly thresholded) associations are raised to `d` before choice, as in `get_perf()`. `NULL` (the default) leaves them unchanged. `xsl_run()` passes a model's `ch_dec` parameter here, so that `bayesian_decay()` scores test trials with the same decision rule it uses elsewhere.

Details

The literal rule is Bayes' rule on the word's row, `P(object | word)` proportional to `m[word, object] * prior(object)` normalized over the objects present. With `pragmatic = TRUE` the objects are instead resolved by a one-step Rational Speech Act pragmatic listener (`rsa_listener()`), which additionally reasons that the speaker could have used a different word – this is what yields *strong* mutual exclusivity when a heard word is lexically ambiguous and a competitor word names one of the candidates. A truly novel (unseen) word carries no lexical evidence and returns the prior in either mode.

Value

A numeric vector of probabilities over `objects`, in the given order, summing to 1.

Examples

```
m <- xsl_run(fgt2009(alpha = 1), get_example_unambiguous_condition())$fits[[1]]$matrix
predict_referent(m, 1, c(1, 2, 3))
predict_referent(m, 1, c(1, 2, 3), prior = c(1, 0, 1)) # object 2 unavailable
```

`propose_but_verify` *Trueswell et al. 2013 propose-but-verify model*

Description

Trueswell et al. 2013 propose-but-verify model

Usage

```
propose_but_verify(alpha, alpha_increase)
```

Arguments

`alpha`	initial association strength
`alpha_increase`	learning rate

Value

An object of class xslMod

Examples

```
mod <- propose_but_verify(alpha = .1, alpha_increase = .5)
xsl_run(mod, get_example_ambiguous_condition())
```

`pursuit` *Stevens et al. 2014 pursuit model*

Description

Stevens et al. 2014 pursuit model

Usage

```
pursuit(gamma, threshold, lambda)
```

Arguments

gamma	learning rate
threshold	threshold (probability) to move an association to the known lexicon
lambda	smoothing probability

Value

An object of class xslMod

Examples

```
mod <- pursuit(gamma = .2, threshold = .3, lambda = .05)
xsl_run(mod, get_example_ambiguous_condition())
```

---

rem *REM (Retrieving Effectively from Memory) model*

Description

Shiffrin & Steyvers' (1997) REM adapted to cross-situational word learning. Word/object identity is a length-w feature vector drawn from a geometric base-rate distribution (g); each trial stores one episodic trace per candidate word-object pairing present, built by imperfectly copying the pair's true features (each feature independently stored with probability u, and if stored, correct with probability c, else a fresh random draw). At test, a candidate pairing's strength is the summed likelihood ratio of every trace given the true (fully-known) probe features, REM's odds-based alternative to `minerva2()`'s dot-product activation.

Usage

```
rem(g, u, c, w = 12)
```

Arguments

g	Base-rate feature-value distribution parameter (larger = more variable/distinctive feature values, in (0, 1))
u	Per-feature storage probability at study, in (0, 1)
c	Probability a stored feature is copied correctly (vs. replaced by a random, possibly-confusable value), in (0, 1)
w	Number of features per item (fixed hyperparameter, not a free cognitive parameter)

Details

Performance note: unlike `minerva2()` (one trace per trial) this model stores one trace per candidate (word, object) pair per trial – on a highly ambiguous corpus (many words/objects per utterance) memory size grows combinatorially rather than linearly with trial count, and both encoding and the `voc_sz * ref_sz` test pass scale with that trace count, making `rem()` the most expensive of this package's memory models on cluttered/naturalistic data. `w` is a fixed representational hyperparameter (not usually treated as a free cognitive parameter), not included by default in this package's DEoptim registry bounds – see `xsl_model_registry()`.

Value

An object of class xslMod

Examples

```
mod <- rem(g = 0.4, u = 0.3, c = 0.7, w = 12)
xsl_run(mod, get_example_ambiguous_condition())
```

---

rescorla_wagner *Rescorla-Wagner (1972) error-driven associative model*

Description

On each trial, the words present are cues and the objects outcomes. Each outcome's prediction is the summed association of the present cues, and its prediction error (`lambda - prediction` if it is present, `0 - prediction` if absent) changes every present cue's association with it by `alpha * beta * lambda` times the error. Cues therefore compete to predict an outcome (e.g. blocking: a word paired with an object that another word already predicts learns little about it). All associations then decay by `C`.

Usage

```
rescorla_wagner(
  C,
  alpha,
  beta,
  lambda,
  outcomes = c("present", "all"),
  cues = c("words", "objects"),
  phi = NULL
)
```

Arguments

`C`	Decay
`alpha`	Salience (fix at 1 unless manipulated)
`beta`	Learning rate – a proportion of lambda

`lambda`	Maximum associative value that a CS can achieve - should be larger than learning rate
`outcomes`	`"present"` (the default) updates only the objects present on a trial; `"all"` also updates absent objects toward 0.
`cues`	`"words"` (the default): words predict objects. `"objects"`: objects predict words.
`phi`	Optional softmax inverse temperature for the choice rule. `NULL` (the default) uses Luce choice.

Details

The defaults (`outcomes = "present"`, `cues = "words"`, Luce choice) are the model as originally applied to cross-situational learning, which learns only about the objects present on a trial. In designs where every pair recurs, it then stays close to counting co-occurrences.

Value

An object of class xslMod

Options

- `outcomes = "all"` follows Rescorla & Wagner (1972) in treating every possible outcome as present (target `lambda`) or absent (target 0) on every trial, so a word's associations with objects that are absent when it is heard weaken. This is discriminative learning (Ramscar et al., 2010; Baayen et al., 2011). Associations can then become negative, which Luce choice treats as 0; `phi` gives a choice rule defined for them.
- `cues = "objects"` reverses the direction of prediction: objects are cues and words outcomes, as in Ramscar et al.'s (2010) feature-label ordering account. The matrix keeps words as rows (`m[w, o]` is object `o`'s association with word `w`), and test choices are still made among a word's row. Cues now compete to predict words rather than objects, so the two directions learn different associations.
- `phi` replaces Luce choice with a softmax choice rule with inverse temperature `phi` (see `softmax_weights()`), applied by `xsl_run()` and to the model's own `perf`.

Choice is invariant to the scale of the associations under Luce choice, and under softmax the scale trades off against `phi`, so `lambda` is not identifiable separately from `beta` (and `phi`): fix it (e.g. at 1) when fitting. Likewise `alpha` only enters as `alpha * beta`.

References

Baayen, R. H., Milin, P., Durdevic, D. F., Hendrix, P., & Marelli, M. (2011). An amorphous model for morphological processing in visual comprehension based on naive discriminative learning. *Psychological Review, 118*(3), 438-481.

Ramscar, M., Yarlett, D., Dye, M., Denny, K., & Thorpe, K. (2010). The effects of feature-label-order and their implications for symbolic learning. *Cognitive Science, 34*(6), 909-957.

Rescorla, R. A., & Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black & W. F. Prokasy (Eds.), *Classical conditioning II* (pp. 64-99). Appleton-Century-Crofts.

Examples

```
mod <- rescorla_wagner(C = 1, alpha = 1, beta = 0.1, lambda = 3)
xsl_run(mod, get_example_ambiguous_condition())
# discriminative learning, with a softmax choice rule
mod <- rescorla_wagner(C = .98, alpha = 1, beta = .3, lambda = 1, outcomes = "all", phi = 5)
xsl_run(mod, get_example_ambiguous_condition())
```

`rollins_corpus` *CHILDES/Rollins naturalistic word-learning corpus (Frank et al. 2009)*

Description

The corpus the intentional Bayesian model of Frank, Goodman & Tenenbaum (2009) was fit to: 619 mother-to-infant utterances from the Rollins corpus in CHILDES, each paired with the set of objects present in the scene (six toys rotated in groups). 416 word types, 22 object types.

Usage

```
rollins_corpus
```

Format

An object of class `list` of length 3.

Details

A list with:

`data` an xslData object with the 619 training utterances (`data$train$words[[t]]` / `data$train$objects[[t]]` are character vectors). No `accuracy` or `test` – there is no human referent-selection data for this corpus.

`gold` the gold-standard lexicon as `list(words, objects)` (34 word-object pairs), for scoring a learned matrix with `get_fscore()`, `get_roc()`, `get_roc_max()`, or `get_tp()`.

`reference` the citation string.

Not part of xsl_datasets (which is scored by SSE against a human accuracy vector these corpora don't have). Imported from the wurwur package (https://github.com/mcfrank/wurwur); see `data-raw/add_wurwur_corpora.R`.

Source

Frank, M. C., Goodman, N. D., & Tenenbaum, J. B. (2009). Using speakers' referential intentions to model early cross-situational word learning. *Psychological Science, 20*(5), 578-585. Corpus originally from Rollins (2003), CHILDES.

Examples

```
m <- suppressWarnings(
  xsl_run(baseline(), rollins_corpus$data)$fits[[1]]$matrix)
get_roc_max(m, gold_lexicon = rollins_corpus$gold)
```

`rsa_listener` *Rational Speech Act pragmatic listener and speaker*

Description

A small, self-contained RSA recursion over a word-by-object meaning matrix (Frank & Goodman, 2012; Smith, Goodman & Frank, 2013), used by `predict_referent()` with `pragmatic = TRUE`. `rsa_listener()` returns `P(object | word)` for a listener who reasons about an informative speaker; `rsa_speaker()` returns `P(word | object)` for that speaker. Both operate on any non-negative `(word x object)` matrix, reading entries as literal meaning weights.

Usage

```
rsa_listener(lex, prior, ...)

rsa_speaker(lex, prior, ...)
```

Arguments

`lex`	A `(word x object)` matrix of literal meaning weights.
`prior`	Numeric prior over objects (one per column of `lex`). Objects with prior 0 are dropped from the reasoning and get listener probability 0.
`...`	Further arguments to the recursion: `alpha` (speaker rationality / softmax temperature, default 3), `cost` (optional per-word production cost vector), and `depth` (pragmatic recursions beyond the literal level; `0` gives the prior-weighted literal listener, `1` the usual one-step pragmatic one).

Value

`rsa_listener()`: a `(word x object)` matrix of `P(object | word)`. `rsa_speaker()`: a `(word x object)` matrix of `P(word | object)`.

Examples

```
# "wug" names either object; a pragmatic listener resolves it to the second
lex <- matrix(c(1, 0, 1, 1), 2, byrow = TRUE)
rsa_listener(lex, c(0.5, 0.5), alpha = 3, depth = 1)
```

show_datasets *Show available datasets in the package*

Description

Returns information about the available cross-situational word learning datasets included in the package.

Usage

```
show_datasets()
```

Value

A data frame with dataset information

Examples

```
show_datasets()
```

show_models *Show available models in the package*

Description

Returns the names of all available cross-situational word learning models, as accepted by `get_group_model_fit()` and `get_crossvalidated_model_fit()`. Most are the package's model functions; a few are variants of them used in the model comparison paper (`uncfam_novelty`, `uncfam_uncertainty_only`, `uncfam_strength`, `fazly_threshold`) or with non-default options and a softmax choice rule (`rescorla_wagner_softmax`, `rescorla_wagner_discriminative`, `kalman_filter_full`).

Usage

```
show_models()
```

Value

A character vector of model names

Examples

```
show_models()
```

softmax_rl *Softmax reinforcement learning model*

Description

A Q-learning-style reinforcement learning model of cross-situational word learning. Unlike every other model in this package, which updates every word-object pair presented together on a trial, this model makes a discrete choice: for each word heard, it samples a single candidate referent from a softmax policy over its current values for *all* known objects (its current best guess at that word's meaning), then updates only that guess's value via a standard TD/delta-rule update, using whether the guess was actually present in the scene as the reward (1 = "confirmed", 0 = "disconfirmed"). This is the only feedback available in a task with no trial-by-trial correct answer, and mirrors the propose-and-verify logic of `propose_but_verify()`/`pursuit()`, but replaces their threshold-based keep/discard rules with graded value learning and an explicit (softmax) exploration/exploitation tradeoff.

Usage

```
softmax_rl(alpha, beta)
```

Arguments

`alpha`	Learning rate
`beta`	Inverse temperature: how deterministically the model favors its highest-valued guess (0 = uniformly random guessing, large = always picks the current best guess)

Value

An object of class xslMod

Examples

```
mod <- softmax_rl(alpha = .3, beta = 3)
xsl_run(mod, get_example_ambiguous_condition())
```

softmax_weights *Softmax choice weights*

Description

Transforms a word-by-object association matrix so that Luce choice over it (as in `get_perf()`, `mafc_test()` and `choice_probabilities()`) is a softmax choice rule with inverse temperature `phi`: a word's probability of choosing object `o` among a set of alternatives is `exp(phi * m[w, o]) / sum(exp(phi * m[w, alternatives]))`. Unlike Luce choice, softmax is defined for negative associations (which error-driven models such as `rescorla_wagner()` and `kalman_filter()` can learn) and gives objects a word has never been paired with nonzero probability.

Usage

```
softmax_weights(m, phi)
```

Arguments

m	A word-by-object association matrix.
phi	Inverse temperature (choice sensitivity), `phi >= 0`: 0 is uniform guessing, and larger values choose the strongest association more deterministically.

Details

`xsl_run()` applies it automatically to a model with a `phi` parameter (e.g. `rescorla_wagner(..., phi = 5)`), to the mean of the simulations' matrices.

Value

A matrix of positive choice weights with `m`'s dimensions and names. Each row is shifted by its maximum before exponentiating, which leaves the choice probabilities unchanged and avoids overflow.

Examples

```
m <- matrix(c(1, -.5, 0, .2), 2, dimnames = list(1:2, 1:2))
get_perf(softmax_weights(m, phi = 3))
```

---

suanda2014 — *Suanda, Mugwanya, & Namy (2014) cross-situational learning in children*

Description

Three conditions (low, medium, and high contextual diversity) from Suanda, Mugwanya, & Namy (2014), *Cross-situational statistical word learning in young children*, Journal of Experimental Child Psychology, 126, 395-411 (doi:10.1016/j.jecp.2014.06.003): 8 word-object pairs, 2 words and 2 objects per training trial, children aged 5-7.

Usage

```
suanda2014
```

Format

A named list of 3 xslData objects, one per condition.

Details

Only each condition's reported mean accuracy is available (no per-item or per-participant data), so `accuracy` is that mean replicated across items. Held out of xsl_datasets (see xsl_holdout_datasets), as in the original bake-off: use it to test whether a model fit elsewhere generalizes, by comparing its mean predicted accuracy with the reported mean.

Examples

```
sapply(suanda2014, \(d) d$accuracy[1])
fit <- xsl_run(uncfam(X = .1, B = 2, C = .97), suanda2014[["Suanda2014-hiCD"]])
mean(fit$fits[[1]]$perf)
```

`tilles` *Tilles and Fontanari (2013) reinforcement model*

Description

Tilles and Fontanari (2013) reinforcement model

Usage

```
tilles(x, b, alpha_0)
```

Arguments

`x`	Reinforcement parameter for stimuli in current context
`b`	Inference parameter, regulating ME and prior info integration (applies either to known words that do not appear in the current context or to new words in the current context)
`alpha_0`	Baseline efficiency corresponding to maximal uncertainty about referent of target word

Value

An object of class xslMod

Examples

```
mod <- tilles(x = 0.6, b = 0.8, alpha_0 = 0.85)
xsl_run(mod, get_example_ambiguous_condition())
```

`todam` *TODAM holographic reduced representation model*

Description

Murdock's (1982) TODAM adapted to cross-situational word learning: word and object vectors (random Gaussian, one per vocabulary item) are bound via circular convolution into a single memory vector `M`, attention- weighted each trial by how familiar (cosine-similar to the current `M`) each word/object already is. At test, a candidate object is circularly correlated back out of `M` and compared (cosine) against each word; similarities are shifted non-negative, row-normalized, and exponentiated by `X` into a graded (Luce-choice-ready) matrix rather than a hard best-match.

Usage

```
todam(X, D = 500)
```

Arguments

X	Choice-rule temperature: how sharply the retrieved-object/word cosine similarity is exponentiated into a choice weight (larger = more winner-take-all)
D	Dimensionality of the random word/object feature vectors (fixed hyperparameter, not a free cognitive parameter)

Details

Performance note: every encoding step and every test comparison requires an FFT-based circular convolution/correlation over a D-length vector, so cost scales with D log(D); unlike minerva2()'s per-trial trace, todam() holds a single fixed-size memory vector regardless of corpus length. D is a fixed representational hyperparameter (not usually treated as a free cognitive parameter), not included by default in this package's DEoptim registry bounds – see xsl_model_registry().

Value

An object of class xslMod

Examples

```
mod <- todam(X = 5, D = 200)
xsl_run(mod, get_example_ambiguous_condition())
```

---

uncfam *Kachergis 2012*

Description

Kachergis et al. 2012 uncertainty- and familiarity-biased associative model

Usage

```
uncfam(X, B, C, variant = c("entropy", "novelty", "uncertainty-only"))
```

Arguments

X	Associative weight to distribute
B	Weighting of uncertainty vs. familiarity
C	Decay
variant	Which variant of the model to fit (one of "entropy", "novelty", "uncertainty-only").

Value

An object of class xslMod

Examples

```
mod <- uncfam(X = .1, C = 1, B = .98)
xsl_run(mod, get_example_ambiguous_condition())

mod <- uncfam(X = .1, C = 1, B = .98, variant = "novelty")
xsl_run(mod, get_example_ambiguous_condition())

mod <- uncfam(X = .1, C = 1, B = 0) # familiarity-only
xsl_run(mod, get_example_ambiguous_condition())

mod <- uncfam(X = .1, C = 1, B = .98, variant = "uncertainty-only")
xsl_run(mod, get_example_ambiguous_condition())
```

---

uncfam_attention *Biased associative model with attention scaled to trial uncertainty*

Description

A variant of uncfam() (Kachergis et al. 2012's uncertainty- and familiarity-biased associative model) in which the associative weight distributed on a trial is additionally scaled by the ratio of that trial's mean object entropy to the mean entropy of all objects seen so far. This implements the "system-level" theory proposed by Fitneva & Christiansen (2015): that learners are more alert (i.e. allocate a higher learning rate) on trials with more uncertain items, such as in a Low Initial Accuracy condition where many items have been mis-paired.

Usage

```
uncfam_attention(X, B, C)
```

Arguments

X	Associative weight to distribute
B	Weighting of uncertainty vs. familiarity
C	Decay

Value

An object of class xslMod

Examples

```
mod <- uncfam_attention(X = .1, C = 1, B = .98)
xsl_run(mod, get_example_ambiguous_condition())
```

`uncfam_elimination` *Biased associative model with learning by elimination*

Description

A variant of `uncfam()` (Kachergis et al. 2012's uncertainty- and familiarity-biased associative model) that adds inference by elimination (mutual exclusivity) to how a trial's associative weight is allocated. If the other words and objects on a trial are already confidently paired with each other, the remaining word and object probably go together, so their pairing should draw attention even though it is still weak. In `uncfam()`, by contrast, an already-strong pair on the trial draws weight *away* from such a pairing.

Usage

```
uncfam_elimination(X, B, C, eps)
```

Arguments

`X`	Associative weight to distribute
`B`	Weighting of uncertainty vs. familiarity
`C`	Decay
`eps`	Weight of elimination inference (0 = identical to `uncfam()`)

Details

On each trial, each on-trial pair (w', o') has a mutual confidence $q(w', o') = P(o'|w')P(w'|o')$, from the row- and column-normalized associations. A pair (w, o) gets elimination support $s(w, o) = \sum_{w' \neq w, o' \neq o} q(w', o')$: how much of the rest of the trial is accounted for by other pairs. The share of the trial's weight X given to (w, o) is then

$$\frac{b(w, o) + \epsilon s(w, o)}{1 + \epsilon \sum s}$$

where $b(w, o)$ is `uncfam()`'s normalized familiarity-and-uncertainty share, so `eps = 0` reproduces `uncfam()` exactly.

Motivated by the initial-accuracy experiment (kachergis_initial_accuracy), where learners were more accurate on an initially mis-paired word the more of its study trials it shared with a word they already knew.

Value

An object of class xslMod

Examples

```
mod <- uncfam_elimination(X = .1, B = .98, C = 1, eps = 1)
xsl_run(mod, get_example_ambiguous_condition())
```

`uncfam_gamma` *Kachergis 2012 with a free familiarity exponent*

Description

A variant of `uncfam()` (Kachergis et al. 2012's uncertainty- and familiarity-biased associative model) that decouples familiarity's contribution to a trial's attentional allocation from uncertainty's. `uncfam()`'s allocation weight for pair (w, o) is `exp(B * entropy) * assocs` – entropy gets a free, tunable exponent (`B`), but familiarity (`assocs`, the pair's current association strength) enters linearly, with an implicit weight of 1 and no free parameter of its own. This adds one parameter, `gamma`, raising familiarity to a free power instead: `assocs^gamma * exp(B * entropy)`. `gamma = 1` is identical to `uncfam()`; `gamma < 1` dampens the "rich get richer" effect (diminishing returns on an already-strong pairing); `gamma > 1` amplifies it (winner-take-all).

Usage

```
uncfam_gamma(X, B, C, gamma)
```

Arguments

`X`	Associative weight to distribute
`B`	Weighting of uncertainty vs. familiarity
`C`	Decay
`gamma`	Familiarity exponent (1 = identical to `uncfam()`)

Details

Cross-validated against all of xsl_datasets (SSE to human 18AFC accuracy, the package's standard model-comparison objective), `uncfam_gamma()` beats plain `uncfam()` in every fold of a 5-fold split (mean test SSE 0.48 vs. 0.56), with a best-fit `gamma` around 0.3 – i.e. familiarity's pull *dampens* with diminishing returns when explaining how well people learn from a passively-received training sequence. A separate analysis fitting this same free-`gamma` substrate to *active* cross-situational word learning (Kachergis, Yu, & Shiffrin's paradigm where learners choose which items to see named next, rather than receiving a fixed passive sequence) found a best-fit `gamma` around 2 instead – amplified, not dampened. Both contexts agree that familiarity should get its own free exponent rather than `uncfam()`'s hard-coded linear one, but disagree on which direction it bends: the process governing moment-to-moment choice of what to study next doesn't appear to be simply the same one governing how much is learned from a given, already-fixed sequence.

Value

An object of class xslMod

Examples

```
mod <- uncfam_gamma(X = .1, C = 1, B = .98, gamma = 1)
xsl_run(mod, get_example_ambiguous_condition())

# gamma = 1 reproduces uncfam() exactly
sse_gamma1 <- xsl_run(uncfam_gamma(X = .1, C = 1, B = .98, gamma = 1),
                      get_example_ambiguous_condition())$sse
sse_uncfam <- xsl_run(uncfam(X = .1, C = 1, B = .98),
                      get_example_ambiguous_condition())$sse
isTRUE(all.equal(sse_gamma1, sse_uncfam))

mod <- uncfam_gamma(X = .1, C = 1, B = .98, gamma = 0.3) # dampened familiarity
xsl_run(mod, get_example_ambiguous_condition())
```

---

uncfam_general *General biased associative model (nests the uncfam family)*

Description

A single biased associative model (Kachergis et al. 2012) whose extensions are free parameters, so that the package's `uncfam()` variants are nested special cases and can be compared by fitting one model with parameters fixed or freed. On each trial, a word-object pair (w, o) on the trial receives a share of the trial's associative weight

$$X_t \cdot \frac{b(w,o) + \epsilon s(w,o)}{1 + \epsilon \sum s}, \quad b(w,o) \propto M_{w,o}^{\gamma} e^{\lambda(U(w)+U(o))}$$

where U is each stimulus's uncertainty (entropy of its associations, or novelty), s is elimination support (see `uncfam_elimination()`), and $X_t = X \cdot (\bar{H}_{trial}/\bar{H}_{all})^{\kappa}$ scales the trial's weight by its objects' relative uncertainty (see `uncfam_attention()`). With finite `K`, attention is sampled rather than allocated: each word attends to `K` objects drawn in proportion to b, and only sampled pairs share the weight (see `uncfam_sampling()`); as `K` grows every pair is sampled and this approaches `K = Inf`.

Usage

```
uncfam_general(
  X,
  B,
  C,
  gamma = 1,
  eps = 0,
  kappa = 0,
  K = Inf,
  uncertainty = c("entropy", "novelty"),
  rho = 0,
  Cs = 0.5
)
```

Arguments

X	Associative weight to distribute
B	Weighting of uncertainty vs. familiarity (λ)
C	Decay
gamma	Familiarity exponent (1 = uncfam())
eps	Weight of learning by elimination (0 = none)
kappa	Exponent on trial-level attention scaling (0 = none, 1 = uncfam_attention())
K	Objects sampled per word per trial (Inf = deterministic allocation; finite = sampled attention, a stochastic model)
uncertainty	Uncertainty measure: "entropy" (default) or "novelty" (1 / (1 + times seen))
rho	Weight of the short-term trace in familiarity (0 = none)
Cs	Per-trial decay of the short-term trace (used only if rho > 0)

Details

A short-term trace (rho > 0) adds recency to the familiarity bias: every update also goes into a second matrix S that decays by Cs per trial (vs. C for M), and familiarity in b becomes $(M + \rho S)^{\gamma}$, so a pairing strengthened on a recent trial draws extra attention. The trace affects only allocation during learning: uncertainty, elimination, and test performance use M.

Special cases (each reproduces the named function exactly):

defaults	uncfam()
B = 0	familiarity only
gamma = 0	uncfam(variant = "uncertainty-only")
uncertainty = "novelty"	uncfam(variant = "novelty")
gamma free	uncfam_gamma()
eps free	uncfam_elimination()
kappa = 1	uncfam_attention()
K finite	uncfam_sampling()

uncfam_predictive() is not nested: its prediction-error update is unnormalized, a different class of rule.

Value

An object of class xslMod

Examples

```
dat <- get_example_ambiguous_condition()
xsl_run(uncfam_general(X = .1, B = .98, C = 1), dat)

# nested comparison: does freeing gamma improve on uncfam()?
xsl_run(uncfam_general(X = .1, B = .98, C = 1, gamma = .5), dat)$sse
```

```
# sampled attention (stochastic)
xsl_run(uncfam_general(X = .1, B = .98, C = 1, K = 1), dat, control = xslControl(n_sim = 50))
```

---

uncfam_predictive *Predictive biased associative model (item-level prediction error)*

Description

A variant of uncfam() (Kachergis et al. 2012's uncertainty- and familiarity-biased associative model) that adds an item-level, Rescorla-Wagner-style prediction error term: on each trial, the amount learned about a word-object association is scaled by how much the word's predicted association strength (summed over the trial's objects) falls short of the maximum value, rather than normalizing to distribute a fixed amount of associative weight across the trial. This lets initially mis-paired ("surprising") items draw more learning than the un-normalized original model allows. Associations are bounded to [0, beta] (with beta = 1): without normalization, the update can otherwise overshoot below 0 (and diverge) or far above beta on a word's first exposure.

Usage

```
uncfam_predictive(X, B, C)
```

Arguments

X	Associative weight to distribute
B	Weighting of uncertainty vs. familiarity
C	Decay

Value

An object of class xslMod

Examples

```
mod <- uncfam_predictive(X = .1, C = 1, B = .98)
xsl_run(mod, get_example_ambiguous_condition())
```

uncfam_sampling *Kachergis 2012*

Description

Kachergis et al. 2012 uncertainty- and familiarity-biased associative model

Usage

```
uncfam_sampling(X, B, C, K = 1)
```

Arguments

X	Associative weight to distribute
B	Weighting of uncertainty vs. familiarity
C	Decay
K	Number of associations to update per word (constraint: B <= X)

Value

An object of class xslMod

Examples

```
mod <- uncfam_sampling(X = .1, C = 1, B = .98, K = 1)
xsl_run(mod, get_example_ambiguous_condition())
```

update_params *Update parameters of xslMod*

Description

Replaces some of xsl_mod's parameter values with params – e.g. a raw numeric vector such as DEoptim's bestmem. The result keeps xsl_mod's original parameter names.

Usage

```
update_params(xsl_mod, params, free = NULL)
```

Arguments

xsl_mod	Object of class xslMod
params	Vector or list of parameter values, in the same order as the leading elements of xsl_mod$params, or as free
free	Optional character vector of parameter names that params replaces, in order

Details

By default `params` replaces the *leading* parameters, matched by position (not by name): `params` may be shorter than `xsl_mod$params`, in which case only the first `length(params)` parameters are replaced. This is what lets `xsl_fit()` optimize a leading subset of a model's parameters (e.g. `uncfam()`'s numeric `X`/`B`/`C` but not its categorical `variant`).

With `free`, `params` instead replaces the named parameters, in the order given – so any subset can be fit while the rest stay fixed (e.g. `rescorla_wagner()`'s `beta`, `C` and `lambda` with `alpha` fixed at 1).

Value

An object of class xslMod

Examples

```
rw <- rescorla_wagner(C = .98, alpha = 1, beta = .5, lambda = 1)
update_params(rw, c(.2, .95, .8), free = c("beta", "C", "lambda"))$params
```

---

vlach_debrock2017 *Vlach & DeBrock (2017) spaced/massed repetition experiment data*

Description

An xslData object from Vlach, H. A., & DeBrock, C. A. (2017). "Remember dax? Relations between children's cross-situational word learning, memory, and language abilities." *Journal of Memory and Language, 93*, 217-230. 12 to-be-learned word-object pairs (symmetric design: word `i` pairs with object `i`), 2 pairs per training trial, 36 trials; pairs vary in whether their 6 repetitions fall on consecutive trials (massed) or are spread out (spaced), the paper's memory/spacing manipulation. `n_subj` is 47 children (ages 2-5).

Usage

```
vlach_debrock2017
```

Format

An object of class `xslData` (inherits from `list`) of length 8.

Details

No per-word accuracy is available from this dataset, only the paper's overall mean across all pairs and participants (accuracy = .5583, sd = .1975; see `data-raw/XSL-dataset-fields.csv`), so `accuracy` is `NA` throughout. **Not included in xsl_datasets** and not scorable by `xsl_run()`'s built-in per-item SSE – usable for running/fitting models and comparing simulated performance against the one real number this paper reports. See `data-raw/add_vlach_debrock2017.R` for construction details.

Examples

```
m <- xsl_run(baseline(), vlach_debrock2017)$fits[[1]]$matrix
dim(m)
```

`xslControl-class` *xslControl S3 class*

Description

Control arguments for `xsl_run()`

Usage

```
xslControl(
  reps = 1,
  start_matrix = NULL,
  test_noise = 0,
  n_sim = 500,
  verbose = FALSE,
  keep_traj = FALSE,
  keep_sims = FALSE,
  objective = c("sse", "binomial", "multinomial"),
  lapse = 0.01
)

new_xslControl(x = list())
```

Arguments

`reps`	Number of times to repeat training
`start_matrix`	Starting matrix
`test_noise`	Test noise
`n_sim`	Number of simulations for stochastic models
`verbose`	Verbosity
`keep_traj`	Whether each model run should record its per-trial association matrix (`xslFit$traj`). Off by default: the trajectory is not used by any function in the package, and for a long corpus it is a serious memory cost (one word-by-object matrix per trial, per simulation). Set `TRUE` only when you want to inspect the learning trajectory yourself.
`keep_sims`	Whether `xsl_run()` should retain the full list of per-simulation `xslFit` objects (`fits[[i]]$sims`). Off by default; `fits[[i]]$responses` (an `n_sim` x n-words matrix of each simulated participant's final per-word accuracy) is always returned instead, at a fraction of the memory.
`objective`	What `xsl_fit()` minimizes, and what `xsl_run()` scores:

- `"sse"` (the default): the `n_subj`-weighted sum of squared errors between model and human accuracy per test item.
- `"binomial"`: the negative log-likelihood of how many participants got each test item right, given the model's probability of a correct choice. Counts come from the condition's `response_matrix`, or otherwise from `accuracy` and `n_subj`.
- `"multinomial"`: the negative log-likelihood of which object each participant chose on each test item, given the model's full choice distribution over the item's alternatives (see `choice_probabilities()`). Needs a `response_matrix`.

The likelihoods treat every response as an independent draw from the model's prediction, ignoring individual differences, so they overstate the evidence for one model over another; rankings are more trustworthy than the absolute size of AIC or likelihood-ratio differences.

`lapse` For the likelihood objectives: the probability that a response is a uniform guess among the item's alternatives rather than the model's choice. Keeps the likelihood finite when a model is certain and a participant chooses otherwise. Fit it by naming `"lapse"` in `xsl_fit()`'s `free`. Ignored by `"sse"`.

`x` List with elements train, test, accuracy, n_subj, label, condition

Value

An object of class xslControl

`xslData-class` *xslData S3 class*

Description

xslData S3 class

Constructor for xslData S3 class

Usage

```
xslData(
  train = list(),
  test = list(),
  accuracy = numeric(),
  n_subj = numeric(),
  label = "",
  condition = "",
  description = "",
  response_matrix = NULL
)

new_xslData(x = list())
```

Arguments

`train`	Training data
`test`	Test data
`accuracy`	Proportion correct per test item: per test trial (in `test` order) if `test` is given, otherwise per word. For a design with several training blocks, each followed by the same test, a matrix with one row per block (see Details)
`n_subj`	Number of subjects
`label`	Label
`condition`	Condition
`description`	Description
`response_matrix`	Optional test-response (confusion) matrix: for each test item (rows, in `accuracy` order) the number of participants choosing each object (columns, one per object in `train`, in sorted label order). For a multi-block design, a list with one such matrix per block
`x`	List with elements train, test, accuracy, n_subj, label, condition, description, response_matrix

Details

Multi-block designs. Some experiments repeat the same training sequence several times, testing after each repetition (a learning trajectory). Give `train` as one block and `accuracy` as a matrix with one row per block; the number of rows is the number of blocks. `xsl_run()` then scores block `b` on a model trained on `b` repetitions of `train`.

Value

An object of class xslData

`xslFit-class` *xslFit S3 class*

Description

xslFit S3 class

Usage

```
xslFit(perf = matrix(), matrix = matrix(), traj = list(), sse = double())
```

Arguments

`perf`	Perf
`matrix`	Matrix
`traj`	Traj
`sse`	SSE

Value

An object of class xslFit

`xsl_datasets` *Cross-situational word learning experiment data*

Description

Cross-situational word learning experiment data

Usage

```
xsl_datasets
```

Format

An object of class `list` of length 47.

Details

Most conditions are scored with `get_perf()` (choice among all objects). Conditions whose test offered only some of the objects carry a `test` specification – for each tested word, the objects offered – and are scored with `mafc_test()`: the four 4AFC conditions (211-214), and the 12 filtering conditions (`filt*E_*L`; Kachergis, Yu, & Shiffrin, 2012). In the filtering conditions each word co-occurs with its own object and its partner's, and was tested twice: with the partner's object absent (items 1-12, own object correct) and with its own object absent (items 13-24, partner's object correct; `test$correct`), so `accuracy` has 24 entries (see `data-raw/fix_filtering_tests.R`). Condition 214's participants were tested with condition 213's 4AFC displays, so the two share a `test` (see `data-raw/fix_4afc_tests.R`).

Every condition also carries a `response_matrix`: for each test item, how many participants chose each object, from which `accuracy` is exactly recoverable (see `data-raw/fix_freqcd_add_responses.R` and `data-raw/raw_responses.R`, which document the source files and each source's participant exclusions).

`xsl_fit` *Fit XSL model using differential evolution*

Description

Fits a model to provided data using the Differential Evolution optimization algorithm. By default it optimizes the model parameters to minimize the sum of squared errors (SSE) between the model's predictions and human accuracy data; with `control = xslControl(objective = "binomial")` (or `"multinomial"`) it instead maximizes the likelihood of the human test responses (see `xslControl()`).

Usage

```
xsl_fit(
  model,
  data,
  lower,
  upper,
  by_data = FALSE,
  control = xslControl(),
 deoptim_control = DEoptim::DEoptim.control(reltol = 0.001, NP = 100, itermax = 100),
  free = NULL,
  n_restarts = 1,
  seed = NULL
)
```

Arguments

model	An object of class xslMod.
data	An object (or list of objects) of class xslData.
lower	Numeric vector of lower bounds for the model's parameters.
upper	Numeric vector of upper bounds for the model's parameters.
by_data	Logical indicating whether to fit to each entry in data separately.
control	Control parameters passed to xsl_run().
deoptim_control	Control parameters passed to DEoptim().
free	Optional names of the parameters that lower/upper bound, in order (see update_params()); by default, the model's leading parameters. With a likelihood objective, "lapse" fits the control's lapse rate along with the model's parameters (e.g. bounds 0 to 1).
n_restarts	Number of independent DEoptim runs per fit; the best (lowest SSE) is kept. Some models' objectives have poor local basins that a single run lands in a good fraction of the time (e.g. uncfam_predictive()), so a few restarts make fits far more reliable.
seed	Optional random seed, set once before fitting, so DEoptim's random initial populations (and a stochastic model's simulations) are reproducible.

Value

A list with one element per fit (one in total, or one per dataset with by_data = TRUE), each the DEoptim result of the best run, which includes the best parameters found (optim$bestmem, named) and their SSE (optim$bestval; the negative log-likelihood with a likelihood objective). With n_restarts > 1, restarts holds every run's best value. With a likelihood objective, each fit also has loglik, n_obs (the number of human responses), k (the number of fitted parameters, including a fitted lapse), and aic and bic.

Examples

```
fit <- xsl_fit(decay(C = .98), xsl_datasets[1:2], lower = .8, upper = 1,
               n_restarts = 2, seed = 1,
           deoptim_control = DEoptim::DEoptim.control(NP = 10, itermax = 5, trace = FALSE))
fit[[1]]$optim$bestmem
fit[[1]]$restarts

# fit by binomial likelihood, with the lapse rate as a free parameter
fit <- xsl_fit(decay(C = .98), xsl_datasets[1:2], lower = c(.8, 0), upper = c(1, .5),
               free = c("C", "lapse"), control = xslControl(objective = "binomial"),
               seed = 1,
           deoptim_control = DEoptim::DEoptim.control(NP = 10, itermax = 5, trace = FALSE))
fit[[1]]$optim$bestmem
fit[[1]]$aic
```

---

xsl_holdout_datasets *Registry of datasets held out from xsl_datasets*

Description

A named list, one entry per dataset in the package that is **not** part of xsl_datasets and therefore never seen by `get_group_model_fit()`/`get_crossvalidated_model_fit()` – i.e. every dataset available for testing a fitted model's generalization to genuinely held-out data. This is a lookup table (dataset name -> how to score it), not a copy of the data itself – look up the dataset by its own name (e.g. `benitez2020`, `vlach_debrock2017`) to use it.

Usage

```
xsl_holdout_datasets
```

Format

An object of class `list` of length 10.

Details

Each entry has:

`scorable` Whether the dataset can be scored with the package's standard diagonal convention (`xsl_run()` + `mafc_test()`/`get_perf()`) out of the box.

`scoring` How to actually score a model's fit against it.

`note` Why it's held out, or any caveat for using it.

Most of these are standalone for a structural reason (partial-`NA` accuracy, a non-square association matrix, or no human accuracy data at all, e.g. the naturalistic corpora) – see each dataset's own `?help` page for the full rationale. `benitez2020`, `kachergis_initial_accuracy`, `kachergis_priming`, `suanda2014`, and `koehne2013` are the exceptions: all are directly scorable the same way `xsl_datasets` conditions are, and are kept out of the group/CV fitting pool specifically to have real held-out data to test generalization against (`suanda2014` and `koehne2013` only at the level of condition means).

Examples

```
xsl_holdout_datasets$benitez2020$scoring
names(xsl_holdout_datasets)[sapply(xsl_holdout_datasets, `[[`, "scorable")]
```

`xsl_run` *Run XSL model*

Description

Run XSL model

Usage

```
xsl_run(model, data, control = xslControl())
```

Arguments

`model`	An object of class xslMod.
`data`	An object (or list of objects) of class xslData.
`control`	Control arguments returned by `xsl_control()`.

Value

A list with `sse`, `unweighted_sse`, and `fits` (one entry per dataset). Each fit has `matrix` (the word-by-object matrix, summed over simulations for a stochastic model), `perf`, `sse`, `data`, `responses` (an `n_sim` x n-words matrix of each simulated participant's final per-word accuracy), and `sims` (the full per-simulation `xslFit` list, `NULL` unless `control` had `keep_sims = TRUE`).

For a multi-block condition (an `xslData` whose `accuracy` has one row per block; see `xslData()`), block `b` is scored on a run trained on `b` repetitions of the training sequence. Its fit's `perf` then has one row per block, `sse` (and `loglik`) sum over blocks, `matrix`, `responses` and `sims` are the last block's, and `blocks` holds each block's own fit. Each block is a separate run, so for a stochastic model the blocks' predictions come from independent simulated learners: right on average, but not one learner's trajectory.

With a likelihood objective (`xslControl(objective = "binomial")` or `"multinomial"`), each fit also has `loglik`, the log-likelihood of that condition's human test responses, and the result has `loglik` (summed over conditions) and `nll` (its negative, which `xsl_fit()` minimizes). Unlike `sse`, which averages conditions weighted by `n_subj`, the likelihood sums over every response, so larger conditions count for more in the same way.

Index

∗ **datasets**
benitez2020, 4
fm_corpus, 13
gangwani2011_category, 14
kachergis2012_highlighting, 23
kachergis_initial_accuracy, 24
kachergis_priming, 24
koehne2013, 27
rollins_corpus, 37
suanda2014, 41
vlach_debrock2017, 51
xsl_datasets, 55
xsl_holdout_datasets, 57

baseline, 3
bayesian_decay, 4
bayesian_decay(), *32*
benitez2020, 4

choice_probabilities, 5
choice_probabilities(), *40*, *53*
create_cooc_matrix, 6
cross_validated_group_fits, 7

decay, 8

fazly, 9
fgt2009, 9
fgt2009(), *11, 12*
fgt2009_rsa, 11
fgt2009_rsa(), *10*, *12*
fgt2009_sweep_alpha, 12
fgt2009_sweep_alpha(), *10*
fm_corpus, *12*, 13

gangwani2011_category, 14
get_crossvalidated_model_fit, 15
get_crossvalidated_model_fit(), *57*
get_example_ambiguous_condition, 16
get_example_unambiguous_condition, 17
get_fscore, 17
get_fscore(), *13*, *37*
get_group_model_fit, 18
get_group_model_fit(), *57*
get_perf, 19
get_perf(), *5, 6*, *32*, *40*, *55*
get_roc, 20
get_roc(), *13*, *37*
get_roc_max *(*get_roc*)*, 20
get_roc_max(), *12*, *37*
get_tp, 21
get_tp(), *13*, *37*
get_train_test_split, 22
guess_and_test, 22

kachergis2012_highlighting, 23
kachergis_initial_accuracy, 24, *45*
kachergis_priming, 24
kalman_filter, 25
kalman_filter(), *19*, *40*
koehne2013, 27

mafc_test, 28
mafc_test(), *6*, *40*, *55*
minerva2, 28
minerva2(), *34, 35*, *43*
multi_sampling, 29

new_xslControl *(*xslControl-class*)*, 52
new_xslData *(*xslData-class*)*, 53
new_xslFit, 30
new_xslMod *(*new_xslFit*)*, 30

plot_roc *(*get_roc*)*, 20
plot_training_trials, 31
predict_referent, 31
predict_referent(), *19*, *28*, *38*
propose_but_verify, 33
propose_but_verify(), *40*
pursuit, 33
pursuit(), *40*

rem, 34
rescorla_wagner, 35
rescorla_wagner(), *19*, *25*, *40*
rollins_corpus, *12, 13*, 37
rsa_listener, 38
rsa_listener(), *32*
rsa_speaker *(*rsa_listener*)*, 38

show_datasets, 39
show_models, 39
softmax_rl, 40
softmax_weights, 40
softmax_weights(), *5*, *26*, *36*
suanda2014, 41

tilles, 42
todam, 42

uncfam, 43
uncfam(), *44–49*
uncfam_attention, 44
uncfam_attention(), *47, 48*
uncfam_elimination, 45
uncfam_elimination(), *47*
uncfam_gamma, 46
uncfam_general, 47
uncfam_predictive, 49
uncfam_predictive(), *48*
uncfam_sampling, 50
uncfam_sampling(), *47*
update_params, 50

vlach_debrock2017, 51

xsl_datasets, *5*, *13, 14*, *23–25*, *27*, *37*, *41*, *46*, *51*, 55, *57*
xsl_fit, 55
xsl_fit(), *52, 53*, *58*
xsl_holdout_datasets, *27*, *41*, 57
xsl_run, 58
xsl_run(), *5, 6*, *24*, *32*, *36*, *41*, *52*, *54*
xslControl *(*xslControl-class*)*, 52
xslControl(), *55*
xslControl-class, 52
xslData, *4*, *13, 14*, *24*, *27*, *37*, *41*, *51*
xslData *(*xslData-class*)*, 53
xslData(), *58*
xslData-class, 53
xslFit *(*xslFit-class*)*, 54
xslFit-class, 54
xslMod *(*new_xslFit*)*, 30
xslMod-class *(*new_xslFit*)*, 30