Package 'UCS'

July 9, 2004

Version 0.3.2

Title The UCS/R libraries

Author Stefan Evert <evert@ims.uni-stuttgart.de>

Maintainer Stefan Evert <evert@ims.uni-stuttgart.de>

Depends R (>= 1.6.0), graphics, stats, boot

Description All libraries from the UCS/R system

License Artistic License or GPL (same terms and conditions as Perl)

URL http://www.collocations.de/

R topics documented:

`add.ranks` *Compute Rankings for Annotated Association Measures (base)*

Description

Add rankings (with or without ties) for specified association measures to a data set object.

Usage

```
add.ranks(ds, keys=ds.find.am(ds), randomise=TRUE, overwrite=TRUE)
```

Arguments

`ds`	a UCS data set object
`keys`	a character vector giving the names of one or more association measures. When it is omitted, rankings are computed for all annotated measures.
`randomise`	if `TRUE`, ties are broken randomly (default). Otherwise, tied rows are assigned the same rank, which is the first free one (as in the Olympic Games). See below for prerequisites.
`overwrite`	if `TRUE`, existing rankings are overwritten (default). Otherwise, association measures for which ranks are already annotated are silently skipped. If you modify association scores within R, be sure to call `add.ranks` with `overwrite=TRUE`.

Details

Since `add.ranks` is based on the `order.by.am` function, the prerequisites are the same: the data set must contain association scores for the `random` measure if `randomise=TRUE` and an `id` variable if `randomise=FALSE`. See the `order.by.am` manpage for further information.

Value

Invisibly returns a copy of `ds` annotated with the requested rankings. The rankings are stored in variables `r.*`, where `*` stands for the name of an association measure (according to the UCS naming conventions, cf. the `am.key2var` manpage).

See Also

`order.by.am`, `am.key2var`, `ds.find.am`, `read.ds.gz`

Examples

```
## from the UCS/R tutorial
GLAW <- read.ds.gz("glaw.scores.ds.gz")
GLAW <- add.ranks(GLAW)

## combine into single command
GLAW <- add.ranks(read.ds.gz("glaw.scores.ds.gz"))
```

`am.key2var` *UCS Variable Names for Association Scores and Rankings (base)*

Description

These functions implement the UCS naming conventions for variables storing association scores and the corresponding ranking. `is.valid.key` checks whether a given string is valid as a name for an association measure. `am.key2var` translates a valid AM name into the corresponding variables (for scores or ranking), and `am.var2key` extracts the AM name from such a variable.

Usage

```
is.valid.key(key, warn=FALSE)

am.key2var(key, rank=FALSE)

am.var2key(var)
```

Arguments

`key`	a character vector, giving the names of one or more association measures
`var`	a character vector of variable names, which must be either association scores or rankings (but both types can be mixed in the vector)
`warn`	if `TRUE`, issues a warning if the vector `key` contains invalid AM names. All invalid entries are listed in the warning message.
`rank`	if `TRUE`, return names of the ranking variables corresponding to the specified association measures. otherwise, return names of variables for association scores.

Value

`is.valid.key` returns a logical vector, `am.var2key` returns a list of AM names ("keys"), and `am.key2var` returns a list of variable names (either for association scores or rankings, depending on the `rank` parameter).

See Also

`builtin.ams` for information about built-in association measures, and the `ucsfile` manpage in UCS/Perl for a description of the UCS naming conventions (enter the shell command `ucsdoc ucsfile`).

Examples

```
am.key2var(c("t.score", "MI"), rank=TRUE)
am.var2key(c("am.t.score", "r.MI"))
```

`binom.conf.interval` *Binomial Confidence Intervals*

Description

Computes confidence intervals for the success probability of a binomial distribution efficiently. Unlike `binom.test`, this function can be applied to vectors.

Usage

```
binom.conf.interval(k, size, limit=c("lower","upper"),
                    conf.level=0.05, one.sided=FALSE)
```

Arguments

`k`	a vector of non-negative integers. Each element represents the number of successes out of `size` trials, i.e. the observed value of a random variable with binomial distribution.
`size`	a vector of positive integers. Each element represents the number of trials of a binomial distribution.
`limit`	if `"upper"`, the upper boundaries of the confidence intervals are returned. If `"lower"`, the lower boundaries are returned. Note that this works both for one-sided and for two-sided confidence intervals.
`conf.level`	the required confidence level, or rather the significance level of the corresponding binomial test (note that this behaviour differs from the built-in `binom.test` function). The default `conf.level=0.05` stands for 95% confidence.
`one.sided`	if `TRUE`, computes one-sided confidence interval (either lower or upper, depending on the value of `limit`). If `FALSE`, a two-sided confidence interval is computed (default).

Details

If `one.sided=TRUE`, the underlying test is one-sided (with alternative `"less"` or `"greater"`, depending on the `limit` parameter), and the non-trivial boundary of the confidence interval is returned.

If `one.sided=FALSE`, the underlying test is two-sided and the requested boundary of the two-sided confidence interval is returned. For efficiency reasons, the `binom.conf.interval` function cheats a little and computes one-sided confidence intervals with significance level `conf.level / 2`.

Value

A numeric vector with the requested boundary of confidence intervals for the unknown success probabilities of binomial variables.

See Also

`binom.test`

`builtin.ams` *UCS/Perl Built-in Association Measures (base)*

Description

`builtin.ams` returns a character vector listing the built-in association measures of the UCS/Perl system (including the standard add-on packages), `is.builtin.am` checks whether a specified measure belongs to this set, and `am.key2desc` returns a short description of the specified measure.

Usage

```
builtin.ams()

is.builtin.am(key)

am.key2desc(key)
```

Arguments

`key`	a character vector specifying the names of one or more association measures

Value

`builtin.ams` returns a character vector containing the names of all built-in association measures, `is.builtin.am` returns a logical vector, and `am.key2desc` returns a character vector with a short description of each of the measures in `key`.

See Also

The information provided by these functions is obtained from the UCS/Perl tool `ucs-list-am`. See the `ucsam` manpage in UCS/Perl for further information about built-in association measures (using the shell command `ucsdoc ucsam`).

Examples

```
print(builtin.ams())
am.key2desc("chi.squared.corr")
```

`Cbeta` *The Beta Function (sfunc)*

Description

Computes the (complete) Beta function and its base 10 logarithm.

Usage

```
Cbeta(a, b, log=FALSE)
```

Arguments

`a, b`	numeric vectors
`log`	if `TRUE`, returns the base 10 logarithm of the Beta function (default: `FALSE`)

Details

This is just a front-end to the built-in `beta` and `lbeta` functions, provided mainly for consistent naming. Note that the logarithmic version is scaled to base 10 logarithms, according to the UCS conventions.

Value

The Beta function with arguments (`a`, `b`), or its base 10 logarithm (if `log=TRUE`).

See Also

`beta`, `Ibeta`, `Rbeta`, `Cgamma`, `Igamma`, `Rgamma`

Examples

```
x <- 5
y <- 3
((x+y+1) * beta(x+1,y+1))^-1 # == choose(x+y, x)
```

`Cgamma` *The Gamma Function (sfunc)*

Description

Computes the (complete) Gamma function and its base 10 logarithm.

Usage

```
Cgamma(a, log=FALSE)
```

Arguments

`a`	a numeric vector
`log`	if `TRUE`, returns the base 10 logarithm of the Gamma function (default: `FALSE`)

Details

This is just a front-end to the built-in `gamma` and `lgamma` functions, provided mainly for consistent naming. Note that the logarithmic version is scaled to base 10 logarithms, according to the UCS conventions.

Value

The Gamma function evaluated at `a`, or its base 10 logarithm (if `log=TRUE`).

See Also

`gamma`, `Igamma`, `Rgamma`, `Cbeta`, `Ibeta`, `Rbeta`

Examples

```
Cgamma(5 + 1) # = factorial(5)
```

`ds.find.am` *List Association Scores and Rankings in Data Set (base)*

Description

`am.in.ds` tests whether a specified association measure is annotated in a data set, `ds.find.am` lists all annotated association measures, and `ds.match.am` searches the data set for AMs whose names may be abbreviated to a unique prefix. All three functions look either for association scores or for rankings.

Usage

```
am.in.ds(ds, keys, rank=FALSE, fail=FALSE)

ds.find.am(ds, rank=FALSE)

ds.match.am(ds, abbrevs, rank=FALSE)
```

Arguments

`ds`	a UCS data set, read from a data set file with the `read.ds.gz` function
`keys`	a character vector of AM names
`abbrevs`	a character vector of AM names, each of which may be abbreviated to a unique prefix (within the data set)
`rank`	if `TRUE`, the functions look for annotated rankings; otherwise, they look for annotated association scores (default)
`fail`	if `TRUE`, the function aborts with an error message unless all specified AMs are annotated in the data set

Details

If any of the abbrevs do not have a unique match in the data set, ds.match.am aborts with an error message (listing all strings that failed to match uniquely).

Value

am.in.ds returns a logical vector of the same length as keys. ds.find.am and ds.match.am return a character vector containing the names of the annotated association measures.

See Also

read.ds.gz, am.var2key

Examples

```
GLAW <- read.ds.gz("glaw.scores.ds.gz")
print(ds.find.am(GLAW))
```

---

evaluation.file *Evaluation Graphs for Association Measures (plots)*

Description

The evaluation.plot function is often invoked twice with the same parameter settings, once for on-screen display, and once for saving the plot to a PostScript file. evaluation.file automates this process, automatically switching between colour mode for the screen version and B/W mode for the PostScript version.

Usage

```
evaluation.file(ds, keys, file, bw=NULL, ...)
```

Arguments

ds	a UCS data set object (passed to evaluation.plot)
keys	a character vector specifying the names of association measures to be evaluated (passed to evaluation.plot)
file	a character string giving the name of a file to which the PostScript version of the plot will be saved
bw	if TRUE, both versions will be in B/W; if FALSE, both versions will be in colour. If unspecified, evaluation.file switches automatically from colour mode (for the screen version) to B/W mode (for the PostScript file), which is the most common use.

Details

PostScript versions can be suppressed by setting

```
ucs.par(do.file=FALSE)
```

In this case, evaluation.file will only draw the screen versions of the graphs, which is convenient when experimenting and while fine-tuning the plots.

See Also

`evaluation.plot`, `ucs.par`, and the tutorial script 'tutorial.R' in the 'script/' directory.

`evaluation.plot` *Evaluation Graphs for Association Measures (plots)*

Description

An implementation of evaluation graphs for the empirical evaluation of association measures in terms of precision and recall, as described in (Evert, 2004, Ch. 5). Graphs of precision, recall and local precision for n-best lists, as well as precision-by-recall graphs are all provided by a single function `evaluation.plot`.

Usage

```
evaluation.plot(ds, keys, tp=ds$b.TP,
                x.min=0, x.max=100, y.min=0, y.max=100,
                x.axis=c("n.best", "proportion", "recall"),
                y.axis=c("precision", "local.precision", "recall"),
                n.first=ucs.par("n.first"), n.step=ucs.par("n.step"),
                cut=NULL, window=400,
                show.baseline=TRUE, show.nbest=NULL, show.npair=NULL,
                conf=FALSE, conf.am=NULL, conf.am2=NULL,
                test=FALSE, test.am1=NULL, test.am2=NULL,
                test.step=ucs.par("test.step"), test.relevant=0,
                usercode=NULL,
                file=NULL, aspect=1, plot.width=6, plot.height=6,
                cex=ucs.par("cex"), lex=ucs.par("lex"), bw=FALSE,
                legend=NULL,
                title=NULL, ...)
```

Arguments

`ds`	a UCS data set object, read in from a data set file with the `read.ds.gz` function. `ds` must contain rankings for the association measures listed in the `keys` parameter (use `add.ranks` to add such rankings to a data set object).
`keys`	a character vector naming up to 10 association measures to be evaluated. Each name may be abbreviated to prefix that must be unique within the measures annotated in `ds`. Use the `ds.find.am` function to obtain a list of measures annotated in the data set, and see the `ucsam` manpage in UCS/Perl for detailed information about the association measures supported by the UCS system (with the shell command `ucsdoc ucsam`).
`tp`	a logical vector indicating true positives, parallel to the rows of the data set `ds`. If `tp` is not specified, the data set must contain a variable named `b.TP` which is used instead.
`x.min, x.max`	the limits of the x-axis in the plot, used to "zoom in" to an interesting region. The interpretation of the values depends on the `x.axis` parameter below. For `x.axis="n.best"` (the default case), `x.min` and `x.max` refer to n-best lists. Otherwise, they refer to percentages ranging from 0 to 100. By default, the full data set is shown.

`y.min, y.max` the limits of the y-axis in the plot, used to "zoom in" to an interesting region. The values are always interpreted as percentages, ranging from 0 to 100. By default, `y.max` is fitted to the evaluation graphs (unless `y.axis="recall"`, where `y.max` is always set to 100).

`x.axis` select variable shown on x-axis. Available choices are the n-best list size n (`"n.best"`, the default), the same as a proportion of the full data set (`"proportion"`), and the recall as a percentage (`"recall"`). The latter produces precision-by-recall graphs. Unless you are silly enough to specify `y.axis="recall"` at the same time, that is.

`y.axis` select variable shown on x-axis. Available choices are the precision (`"precision"`, the default), an estimate for local precision (`"local.precision"`, see details below), and the recall (`"recall"`). All three variables are shown as percentages ranging from 0 to 100.

`n.first` the smallest n-best list to be evaluated. Shorter n-best lists typically lead to highly unstable evaluation graphs. The standard setting is 100, but a higher value may be necessary for random sample evaluation (see details below). If `n.first` is not specified, the default supplied by `ucs.par` is used.

`n.step` the step width for n-best lists in the evaluation graphs. Initially, precision and recall are computed for all n-best lists, but only every `n.step`-th one is plotted, yielding graphs that look less jagged and reducing the size of generated PostScript files (see the `file` parameter below). If `n.step` is not specified, the default supplied by `ucs.par` is used.

`cut` for each association measure, pretend that the data set consists only of the `cut` highest-ranked candidates according to this measure. This trick can be used to perform an evaluation of n-best lists without having to annotate the full data set. The candidates from all relevant n-best lists are combined into a single data set file and `cut` is set to n.

`window` number of candidates to consider when estimating local precision (default: 400), i.e. with the option `y.axis="local"`. Values below 400 or above 1000 are rarely useful. See below for details.

`show.baseline` if `TRUE`, show baseline precision as dotted horizontal line with label (this is the default). Not available when `y.axis="recall"`.

`show.nbest` integer vector of n-best lists that will be indicated as thin vertical lines in the plot. When `x.axis="recall"`, the n-best lists are shown as diagonal lines.

`show.npair` when `x.axis="proportion"`, the total number of candidates in `ds` is shown in the x-axis label. Set `show.npair=NULL` to suppress this, or set it to an integer value in order to lie about the number of candidates (rarely useful).

`conf` if `TRUE`, confidence intervals are shown as coloured or shaded regions around one or two precision graphs. In this case, the parameter `conf.am` must also be specified. Alternatively, `conf` can be set to a number indicating the significance level to be used for the confidence intervals (default: 0.05, corresponding to 95% confidence). See below for details. Note that `conf` is only available when `y.axis="precision"`.

`conf.am` name of the association measure for which confidence intervals are displayed (may be abbreviated to a prefix that is unique within `keys`)

`conf.am2` optional second association measure, for which confidence intervals will also be shown

`test` if `TRUE`, significance tests are carried out for the differences between the evaluation results of two association measures, given as `test.am1` and `test.am2` below. Alternatively, `test` can be set to a number indicating the significance level to be used for the tests (default: 0.05). n-best lists where the result difference is significant are indicated by arrows between the respective evaluation graphs (when `x.axis="recall"`) or by coloured triangles (otherwise). See details below. Note that `test` is *not* available when `y.axsis="local"`.

`test.am1` the first association measure for significance tests (may be abbreviated to a prefix that is unique within `keys`). Usually, this is the measure that achieves better performance (but tests are always two-sided).

`test.am2` the second association measure for significance tests (may be abbreviated to a prefix that is unique within `keys`)

`test.step` the step width for n-best lists where significance tests are carried out, as a multiple of `n.step`. The standard setting is 10 since the significance tests are based on the computationally expensive `fisher.test` functio and since the triangles or arrows shown in the plot are fairly large. If `test.step` is not specified, the default supplied by `ucs.par` is used.

`test.relevant` a positive number, indicating the estimated precision differences that are considered "relevant" and that are marked by dark triangles or arrows in the plot. See below for details.

`usercode` a callback function that is invoked when the plot has been completed, but before the legend box is drawn. This feature is mainly used to add something to a plot that is written to a PostScript file. The `usercode` function is invoked with parameters `region=c(x.min,x.max,y.min,y.max)` and `pr`, a list of precision/recall tables (as returned by `precision.recall`) for each of the measures in `keys`.

`file` a character string giving the name of a PostScript file. If specified, the evaluation plot will be saved to `file` rather than displayed on screen. See `evaluation.file` for a function that combines both operations.

`aspect` a positive number specifying the desired aspect of the plot region (only available for PostScript files). In the default case `x.axis="n.best"`, `aspect` refers to the absolute size of the plot region. Otherwise, it specifies the size ratio between percentage points on the x-axis and the y-axis. Setting `aspect` modifies the height of the plot (`plot.height`).

`plot.width, plot.height`
the width and height of a plot that is written to a PostScript file, measured in inches. `plot.height` may be overridden by the `aspect` parameter, even if it is set explicitly.

`cex` character expansion factor for labels, annotations, and symbols in the plot (see `par` for details). If `cex` is not specified, the default supplied by `ucs.par` is used.

`lex` added to the line widths of evaluation graphs and some decorations (note that this is not an expansion factor). If `lex` is not specified, the default supplied by `ucs.par` is used.

`bw` if `TRUE`, the evaluation plot is drawn in black and white, which is mostly used in conjunction with `file` to produce figures for articles (defaults to `FALSE`). See below for details.

`legend`	a vector of character strings or expressions, used as labels in the legend of the plot (e.g. to show mathematical symbols instead of the names of association measures). Use `legend=NULL` to suppress the display of a legend box.
`title`	a character vector or expression to be used as the main title of the plot (optional)
`...`	any other arguments are set as local graphics parameters (using `par`) before the evaluation plot is drawn

Details

When `y.axis="local.precision"`, the evaluation graphs show **local precision**, i.e. an estimate for the density of true positives around the n-th rank according to the respective association measure. Local precision is smoothed using a kernel density estimate with a Gaussian kernel (from the `density` function), based on a symmetric window covering approximately `window` candidates (default: 400). Consequently, the resulting values do not have a clear-cut interpretation and should not be used to evaluate the performance of association measures. They are rather a means of exploratory data analysis, helping to visualise the relation between association scores and the true positives in a data set (see Evert, 2004, Sec. 5.2 for an example).

In order to generalise evaluation results beyond the specific data set on which they were obtained, it is necessary to compute confidence intervals for the observed precision values and to test whether the observed result differences are significant. See (Evert, 2004, Sec. 5.3) for the methods used and the interpretation of their results.

Confidence intervals are computed by setting `conf=TRUE` and selecting an association measure with the `conf.am` parameter. The confidence intervals are displayed as a coloured or shaded region around the precision graph of this measure (confidence intervals are not available for graphs of recall or local precision). The default confidence level of 95% will rarely need to be changed. Optionally, a second confidence region can be displayed for a measure selected with the `conf.am2` parameter.

Significance tests for the result differences are activated by setting `test=TRUE` (not available for graphs of local precision). The evaluation results of two association measures (specified with `test.am1` and `test.am2`) are compared for selected n-best lists, and significant differences are marked by coloured triangles or arrows (when `x.axis="recall"`). The default significance level of 0.05 will rarely need to be changed. Use the `test.step` parameter to control the spacing of the triangles or arrows.

A significant difference indicates that measure A is truly better than measure B, rather than just as a coincidence in a single evaluation experiment. Formally, this "true performance" can be defined as the average precision of a measure, obtained by averaging over many similar evaluation experiments. Thus, a significant difference means that the average precision of A is higher than that of B, but it does not indicate how great the difference is. A tiny difference (say, of half a percent point) is hardly **relevant** for an application, even if there is significant evidence for it. If the `test.relevant` parameter is set, the `evaluation.plot` function attempts to estimate whether there is significant evidence for a relevant difference (of at least a many percent points as given by the value of `test.relevant`), and marks such cases by darker triangles or arrows. This feature should be considered experimental and used with caution, as the computation involves many approximations and guesses (exact statistical inference for the difference in true precision not being available).

It goes without saying that confidence regions and significance tests do not allow evaluation results to be generalised to a different extraction task (i.e. another type of cooccurrences or another definition of true positives), or even to the same task under different conditions

(such as a source corpus from a different domain, register, time, or a corpus of different size). The unpredictability of the performance of association measures for different extraction tasks or under different conditions has been confirmed by various evaluation studies.

Generally, evaluation plots can be drawn in two modes: **colour** (`bw=FALSE`, the default) or **black and white** (`bw=TRUE`). The styles of evaluation graphs are controlled by the respective settings in `ucs.par`, while the appearance of various other elements is hard-coded in the `evaluation.plot` function. In particular, confidence regions are either filled with a light background colour (colour mode) or shaded with diagonal lines (B/W mode). The triangles or arrows used to mark significant differences are yellow or red (indicating relevance) in colour mode, and light grey or dark grey (indicating relevance) in B/W mode. B/W mode is mainly used to produce PostScript files to be included as figures in articles, but can also be displayed on-screen for testing purposes.

The `evaluation.plot` function supports **evaluation based on random samples**, or RSE for short (Evert, 2004, Sec. 5.4). Missing values (`NA`) in the `tp` vector (or the `b.TP` variable in `ds`) are interpreted as unannotated candidates. In this case, precision, recall and local precision are computed as maxmium-likelihood estimates based on the annotated candidates. Confidence intervals and significance tests, which should not be absent from any RSE, are adjusted accordingly. A confidence interval for the baseline precision is automatically shown (by thin dotted lines) when RSE is detected. Note that n-best lists (as shown on the x-axis) still refer to the full data set, not just to the number of annotated candidates.

Note

The following functions are provided for compatibility with earlier versions of UCS/R: `precision.plot`, `recall.plot`, and `recall.precision.plot`. They are simple front-ends to `evaluation.plot` with the implicit parameter settings `y.axis="recall"` and `y.axis="precision", x.axis="recall"` for the latter two.

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`ucs.par`, `evaluation.file`, `read.ds.gz`, and `precision.recall`. The R script 'tutorial.R' in the 'script/' directory provides a gentle introduction to the wide range of possibilities offered by the `evaluation.plot` function.

`EVm` *Expected Frequency Spectrum of a LNRE Model (zm, fzm)*

Description

Computes the expected frequency spectrum, relative frequency spectrum, and conditional parameter distribution of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
EVm(model, m, N, rho=1, relative=FALSE, ratio=FALSE)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
`m`	a vector of positive integers, representing frequency ranks
`N`	a vector of positive integers, representing sample sizes; either `m` or `N` should be a single number
`rho`	a vector of numbers in the range $[0, 1]$. If `length(rho) > 1`, both `m` and `N` should be single numbers. See below for details.
`relative`	if `TRUE`, computes the relative frequency spectrum (see below for details)
`ratio`	if `TRUE`, computes the ratio between consecutive elements in the expected frequency spectrum

Details

The expected frequency spectrum consists of the numbers $E[V_m(N)]$, which stand for the expected number of types in frequency class m at sample size N, according to the LNRE model `model` (see Baayen, 2001).

If `relative=TRUE`, the relative frequency spectrum $E[V_m(N)]/E[V(N)]$ is returned. If `ratio=TRUE`, the ratios between consecutive expected class sizes, $E[V_{m+1}(N)]/E[V_m(N)]$, are returned.

If `rho` is specified, the conditional parameter distribution $E[V_{\rho,m}(N)]$ is returned, i.e. the expected number of types in frequency class m at sample size N with probability parameter $\pi \leq \rho$ (Evert, 2004, Ch. 4). For `relative=TRUE`, the expected proportion $E[V_{\rho,m}(N)]/E[V(N)]$ is returned instead.

Value

a numeric vector of appropriate length (determined either by `m`, `N`, or `rho`)

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`zm`, `fzm`, `EVm`

EV — *Expected Vocabulary Size of a LNRE Model (zm, fzm)*

Description

Computes the expected vocabulary size of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
EV(model, N)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
`N`	a vector of positive integers, representing sample sizes

Details

The expected vocabulary size $E[V(N)]$ is the expected number of types at sample size N, according to the LNRE model `model` (see Baayen, 2001).

Value

a numeric vector of the same length as `N`

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`, `EVm`

`fzm` *The Finite Zipf-Mandelbrot LNRE Model (fzm)*

Description

Object constructor for a finite Zipf-Mandelbrot (fZM) LNRE model with parameters α, A and B (see Evert, 2004a for details). Either the parameters are specified explicitly, or one or more of them can be estimated from an observed frequency spectrum.

Usage

```
fzm(alpha, A, B)

fzm(alpha, A, N, V)

fzm(alpha, N, V, spc, m.max=15, stepmax=10, debug=FALSE)

fzm(N, V, spc, m.max=15, stepmax=10, debug=FALSE)
```

Arguments

`alpha`	a number in the range $(0, 1)$, the shape parameter α of the fZM model. `alpha` can automatically be estimated from `N`, `V`, and `spc`.
`A`	a small positive number $A \ll 1$, the parameter A of the fZM model. `A` can automatically be estimated from `N`, `V`, and `spc`.
`B`	a large positive number $B \gg 1$, the parameter B of the fZM model. `B` can automatically be estimated from `N` and `V`.
`N`	the sample size, i.e. number of observed tokens

`V`	the vocabulary size, i.e. the number of observed types
`spc`	a vector of non-negative integers representing the class sizes V_m of the observed frequency spectrum. The vector is usually read from a file in `lexstats` format with the `read.spectrum` function.
`m.max`	the number of ranks from `spc` that will be used to estimate the α parameter
`stepmax`	maximal step size of the `nlm` function used for parameter estimation. It should not be necessary to change the default value.
`debug`	if `TRUE`, print debugging information during the parameter estimation process. This feature can be useful to find out why parameter estimation fails.

Details

The fZM model with parameters $\alpha \in (0, 1)$ and $C > 0$ is defined by the type density function

$$g(\pi) := C \cdot \pi^{-\alpha-1}$$

for $A \leq \pi \leq B$. The normalisation constant C is determined from the other parameters by the condition

$$\int_A^B \pi \cdot g(\pi) d\pi = 1$$

The parameters α and A are estimated simultaneously by nonlinear minimisation (`nlm`) of a multinomial chi-squared statistic for the observed against the expected frequency spectrum. Note that this is different from the multivariate chi-squared test used to measure the goodness-of-fit of the final model (Baayen, 2001, Sec. 3.3).

Value

An object of class `"fzm"` with the following components:

`alpha`	value of the α parameter
`A`	value of the A parameter
`B`	value of the B parameter
`C`	value of the normalisation constant C
`N`	number of observed tokens (if specified)
`V`	number of observed types (if specified)
`spc`	observed frequency spectrum (if specified)

This object `print`s a short summary, including the population size S and a comparison of the first ranks of the observed and expected frequency spectrum (if available).

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

See Also

`zm`, `EV`, `EVm`, `write.lexstats`, `read.spectrum`, and `spectrum.plot`

`iaa.kappa` *Inter-Annotator Agreement: Cohen's Kappa (iaa)*

Description

Compute the kappa statistic (Cohen, 1960) as a measure of intercoder agreement on a binary variable between two annotators, as well as a confidence interval according to Fleiss, Cohen & Everitt (1969). The data can either be given in the form of a 2×2 contingency table or as two parallel annotation vectors.

Usage

```
iaa.kappa(x, y=NULL, conf.level=0.95)
```

Arguments

`x`	either a 2×2 contingency table in matrix form, or a vector of logicals
`y`	a vector of logicals; ignored if `x` is a matrix
`conf.level`	confidence level of the returned confidence interval (default: 0.95, corresponding to 95% confidence)

Value

A data frame with a single row and the following variables:

`kappa`	sample estimate for the kappa statistic
`sd`	sample estimate for the standard deviation of the kappa statistic
`kappa.min, kappa.max`	two-sided asymptotic confidence interval for the "true" kappa, based on normal approximation with estimated variance

The single-row data frame was chosen as a return structure because it `print`s nicely, and results from different comparisons can easily be combined with `rbind`.

References

Cohen, Jacob (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**, 37–46.

Fleiss, Joseph L.; Cohen, Jacob; Everitt, B. S. (1969). Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin*, **72**(5), 323–327.

See Also

`iaa.pta`

Examples

```
## kappa should be close to zero for random codings
p <- 0.1                        # proportion of true positives
x <- runif(1000) < p            # 1000 candidates annotated randomly
y <- runif(1000) < p
iaa.kappa(x, y)
```

`iaa.pta` *Inter-Annotator Agreement: Estimates for the Proportion of True Agreement (iaa)*

Description

Compute confidence interval estimates for the proportion of true agreement between two annotators on a binary variable, as described by Krenn, Evert & Zinsmeister (2004). `iaa.pta.conservative` computes a conservative estimate that is rarely useful, while `iaa.pta.homogeneous` relies on additional assumptions. The data can either be given in the form of a 2×2 contingency table or as two parallel annotation vectors.

Usage

```
iaa.pta.conservative(x, y=NULL, conf.level=0.95, debug=FALSE)

iaa.pta.homogeneous(x, y=NULL, conf.level=0.95, debug=FALSE)
```

Arguments

`x`	either a 2×2 contingency table in matrix form, or a vector of logicals
`y`	a vector of logicals; ignored if `x` is a matrix
`conf.level`	confidence level of the returned confidence interval (default: 0.95, corresponding to 95% confidence)
`debug`	if `TRUE`, show which divisions of the data are considered when computing the confidence interval (see Krenn, Evert & Zinsmeister, 2004)

Details

This approach to measuring intercoder agreement is based on the assumption that the observed **surface agreement** in the data can be divided into **true agreement** (i.e. candidates where both annotators make the same choice *for the same reasons*) and **chance agreement** (i.e. candidates on which the annotators agree purely by coincidence). The goal is to estimate the proportion of candidates for which there is true agreement between the annotators, referred to as PTA.

The two functions differ in how they compute this estimate. `iaa.pta.conservative` considers all possible divisions of the observed data into true and chance agreement, leading to a conservative confidence interval. This interval is almost always too large to be of any practical value.

`iaa.pta.homogeneous` makes the additional assumption that the average proportion of true positives is the same for the part of the data where the annotators reach true agreement and for the part where they agree only by chance. Note that there is no *a priori* reason why this should be the case. Interestingly, the confidence intervals obtained in this way for the PTA correspond closely to those for Cohen's kappa statistic (`iaa.kappa`).

Value

A numeric vector giving the lower and upper bound of a confidence interval for the proportion of true agreement (both in the range $[0, 1]$).

Note

`iaa.pta.conservative` is a computationally expensive operation based on Fisher's exact test. (It doesn't use `fisher.test`, though. If it did, it would be even slower than it is now.) In most circumstances, you will want to use `iaa.pta.homogeneous` instead.

References

Krenn, Brigitte; Evert, Stefan; Zinsmeister, Heike (2004). Determining intercoder agreement for a collocation identification task. In preparation.

See Also

`iaa.kappa`

Examples

```
## how well do the confidence intervals match the true PTA?
true.agreement <- 700           # 700 cases of true agreement
chance <- 300                   # 300 cases where annotations are independent
p <- 0.1                        # average proportion of true positives
z <- runif(true.agreement) < p  # candidates with true agreement
x.r <- runif(chance) < p        # randomly annotated candidates
y.r <- runif(chance) < p
x <- c(z, x.r)
y <- c(z, y.r)
cat("True PTA =", true.agreement / (true.agreement + chance), "\n")
iaa.pta.conservative(x, y)      # conservative estimate
iaa.pta.homogeneous(x, y)       # estimate with homogeneity assumption
```

`Ibeta` *The Incomplete Beta Function (sfunc)*

Description

Computes the incomplete Beta function and its inverse. The Beta value can be scaled to a base 10 logarithm.

Usage

```
Ibeta(x, a, b, log=FALSE)

Ibeta.inv(y, a, b, log=FALSE)
```

Arguments

`a, b`	non-negative numeric vectors, the parameters of the incomplete Beta function
`x`	a numeric vector with values in the range $[0, 1]$, the point at which the incomplete Beta function is evaluated
`y`	a numeric vector, the values of the incomplete Beta function (or their base 10 logarithms if `log=TRUE`)
`log`	if `TRUE`, the Beta values are base 10 logarithms (default: `FALSE`)

Details

The incomplete Beta function is defined by the Beta integral

$$B(x;a,b) = \int_0^x t^{a-1}(1-t)^{b-1}\,dt$$

Value

`Ibeta` returns the incomplete Beta function with parameters (`a`,`b`) evaluated at point `x`.

`Ibeta.inv` returns the point `x` at which the incomplete Beta function with parameters (`a`,`b`) evaluates to `y`.

See Also

`Cgamma`, `Igamma`, `Rgamma`, `Cbeta`, `Rbeta`

`Igamma` *The Incomplete Gamma Function (sfunc)*

Description

Computes the incomplete Gamma function and its inverse. Both the lower and the upper incomplete Gamma function are supported, and the Gamma value can be scaled to a base 10 logarithm.

Usage

```
Igamma(a, x, lower=TRUE, log=FALSE)

Igamma.inv(a, y, lower=TRUE, log=FALSE)
```

Arguments

`a`	a non-negative numeric vector, the parameter of the incomplete Gamma function
`x`	a non-negative numeric vector, the point at which the incomplete Gamma function is evaluated
`y`	a numeric vector, the values of the incomplete Gamma function (or their base 10 logarithms if `log=TRUE`)
`lower`	if `TRUE`, computes the lower incomplete Gamma function (default). Otherwise, computes the upper incomplete Gamma function.
`log`	if `TRUE`, the Gamma values are base 10 logarithms (default: `FALSE`)

Details

The upper incomplete Gamma function is defined by the Gamma integral

$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}\,dt$$

The lower incomplete Gamma function is defined by the complementary Gamma integral

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}\,dt$$

Value

`Igamma` returns the (lower or upper) incomplete Gamma function with parameter `a` evaluated at point `x`.

`Igamma.inv` returns the point `x` at which the (lower or upper) incomplete Gamma function with parameter `a` evaluates to `y`.

See Also

`Cgamma`, `Rgamma`, `Cbeta`, `Ibeta`, `Rbeta`

`lnre.goodness.of.fit` *Perform Goodness-of-Fit Evaluation of LNRE Model*

Description

Uses the external `lnreChi2` program from the `lexstats` package to evaluate the goodness-of-fit of a LNRE model with a multivariate chi-squared test (Baayen, 2001, Sec. 3.3).

Usage

```
lnre.goodness.of.fit(model, m.max=Inf, debug=FALSE)
```

Arguments

`model`	an object representing a LNRE model whose parameters have been estimated from observed word frequency data. The model must provide the `write.lexstats` method, which creates the necessary data files. Currently, the Zipf-Mandelbrot (ZM, class `"zm"`) and the finite Zipf-Mandelbrot (fZM, class `"fzm"`) model are supported.
`m.max`	highest frequency rank to be included in the evaluation (limited by the number of ranks that the `write.lexstats` method saves to disk, currently 15)
`debug`	if `TRUE`, displays output of the `lnreChi2` program and does not delete temporary data files

Details

This function relies on the availability of the external `lnreChi2` program from the `lexstats` package, which must be in the user's search path. It uses the `write.lexstats` function to create the necessary data files in a temporary directory, invokes the `lnreChi2` tool, and parses its report file.

All LNRE models that implement a compatible `write.lexstats` method are supported. Currently, these are objects of class `"zm"` or `"fzm"`. The object must include observed word frequency data (in components `N`, `V`, and `spc`), which is usually achieved by estimating the model parameters from the observed frequency spectrum.

Value

A data frame with one row and three columns:

`X2`	the value of the multi-variate χ^2 test statistic
`df`	the degrees of freedom of the approximate χ^2 distribution of the test statistic under the null hypothesis
`p`	the p-value for the test

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`, `write.lexstats`

`order.by.am` *Sort Rows of a Data Set by Association Scores (base)*

Description

Sort the rows of a data set according to the annotated scores of an association measure (in descending order). Ties in the ordering are broken randomly by default, using the `random` association measure to yield a reproducible ordering.

Usage

```
order.by.am(ds, am, randomise=TRUE)
```

Details

With `randomise=TRUE`, the data set must contain a variable named `am.random`, which is used to break ties in the ordering. Otherwise, tied rows are arranged according to their ID values, and the corresponding `id` variable must be annotated in the data set.

The `random` association measure is used for breaking ties (rather than random numbers generated on the fly) in order to ensure that the ordering is reproducible. If this measure has not been annotated in a data set file, you can easily add the required variable to a data set `ds` with the command

```
ds\$am.random <- runif(nrow(ds))
```

You should probably use `set.seed` to ensure a reproducible ordering.

Value

an integer vector of row numbers, which can be used as a row index for the data set object

See Also

`read.ds.gz`, `add.ranks`

`precision.recall` *Compute Precision and Recall for N-Best Lists (base)*

Description

Computes precision and recall of n-best lists for a UCS data set annotated with true positives and rankings (based on association scores). This function forms the basis for the evaluation graphs in the `plots` packages.

Usage

```
precision.recall(ds, am, tp=ds$b.TP, step=1, first=1, cut=0, window=0)
```

Arguments

`ds`	a UCS data set object
`am`	a character string giving the name of an association measure. The corresponding ranking must be annotated in the data set (usually with the `add.ranks` function).
`tp`	a logical vector, which must be parallel to the rows of the data set. `TRUE` values indicate true positives (see details below for the use of missing values). If `tp` is omitted, the data set must contain a Boolean variable `b.TP` which is used instead.
`step`	step width for n-best lists considered, i.e. precision and recall are computed for every `step`-th value of n only (default: 1)
`first`	smallest n-best list for which precision and recall are computed (default: 1)
`cut`	pretend that data set consists only of the first `cut` rows in the ranking, i.e. treat `cut`-best list as full data set (for percentage and recall).
`window`	if specified, local precision is estimated, considering a window of approximately the given size around each value of n (uses the `density` function for smoothing). Useful window sizes range from 400 to 1000.

Details

The `precision.recall` function supports evaluation based on random samples (cf. Evert, 2004, Sec. 5.4). Any `NA` values in the `tp` parameter (or the `b.TP` variable) are interpreted as unannotated candidates. Precision and recall values are computed from the annotated candidates only (as are the `tp`, `fp`, and `lp` variables in the returned data frame). For a random sample evaluation, confidence intervals should always be supplied with the raw precision values, and result differences should be tested for significance. Such tests are implemented by the `evaluation.plot` function, for instance.

Value

An invisible data frame with rows corresponding to n-best lists and the following variables:

`n`	the number of candidates in the n-best list
`perc`	the same as a percentage of the full data set (or the `cut` highest-ranking candidates if specified)

`tp`	the number of true positives in the n-best list
`fp`	the number of false positives in the n-best list
`precision`	the precision of the n-best list, i.e. the number of TPs divided by n
`recall`	the recall of the n-best list, i.e. the number of TPs divided by the total number of TPs in the data set
`lp`	if `window` is specified, an estimate for the *local precision*, i.e. the density of TPs in the vicinity of the n-th rank. Averages over a symmetric window of approximately the specified total size by convolution with a Gaussian kernel (using the `density` function).

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`add.ranks`, `read.ds.gz`, `evaluation.plot`

`Rbeta` *The Regularized Beta Function (sfunc)*

Description

Computes the regularized Beta function and its inverse. The Beta value can be scaled to a base 10 logarithm.

Usage

```
Rbeta(x, a, b, log=FALSE)

Rbeta.inv(y, a, b, log=FALSE)
```

Arguments

`a, b`	non-negative numeric vectors, the parameters of the regularized Beta function
`x`	a numeric vector with values in the range $[0, 1]$, the point at which the regularized Beta function is evaluated
`y`	a numeric vector, the values of the regularized Beta function (or their base 10 logarithms if `log=TRUE`)
`log`	if `TRUE`, the Beta values are base 10 logarithms (default: `FALSE`)

Details

The regularized Beta function scales the incomplete Beta function to the interval $[0, 1]$, by dividing through $B(a, b)$, i.e.

$$I(x; a, b) = \frac{B(x; a, b)}{B(a, b)}$$

Value

`Rbeta` returns the regularized Beta function with parameters (`a`,`b`) evaluated at point `x`.

`Rbeta.inv` returns the point `x` at which the regularized Beta function with parameters (`a`,`b`) evaluates to `y`.

See Also

`Cgamma`, `Igamma`, `Rgamma`, `Cbeta`, `Ibeta`

`read.ds.gz` *Load UCS data set file (base)*

Description

Load a UCS data set file, which is uncompressed on the fly if necessary.

Usage

```
read.ds.gz(filename)
```

Arguments

`filename`	name, partial or full path of the data set file to be loaded.

Details

When the specified file is not found in the current directory, it is automatically searched in the standard UCS data library (the 'DataSet/' directory and its subdirectories). Should there be multiple matches, a warning is issued and the first match is used. You may specify partial paths to identify the desired file unambiguously (e.g. `"Distrib/dickens.ds.gz"`). The automatic search facility is suppressed when `filename` is an explicit absolute or relative path (starting with `/` or `./`).

`gzip`-compressed data set files, whose name must end in `.gz`, are automagically decompressed.

Value

A data frame with column names (i.e. variables) corresponding to those in the data set file. `l1` and `l2` are read as character vectors, all other string variables (`f.*`) are converted into factors, and Boolean variables (`b.*`) are converted into logicals.

Any comments and global variables in the file header are discarded.

Examples

```
## load GLAW data set from UCS distribution
GLAW <- read.ds.gz("glaw.ds.gz")
```

`read.spectrum` *Read Frequency Spectrum File (lexstats)*

Description

Read a word frequency spectrum from a `.spc` file in `lexstats` format (see Baayen, 2001). Returns spectrum as integer vector, possibly including zeroes, whose m-th element gives the number of types V_m with frequency rank m. Also computes sample size N and vocabulary size V.

Usage

```
read.spectrum(file, m.max=Inf, expected=FALSE)
```

Arguments

`file`	a character string giving the name of a frequency spectrum file in `lexstats` format (usually with the extension `.spc`)
`m.max`	maximum length of frequency spectrum, i.e. frequency ranks $m > m_{\max}$ are discarded. Setting `m.max` is a good idea if there are high-frequency types, so that the spectrum is sparse. For most applications, only the first 10 to 100 ranks are of interest.
`expected`	if `TRUE`, reads expected class sizes (in the `EVm` column) rather than the observed ones (in the `Vm` column). This is only possible when the `.spc` file was generated by a LNRE model, of course.

Value

A list with the following components:

`spc`	an integer vector containing the class sizes V_m
`N`	the sample size computed from the spectrum
`V`	the vocabulary size computed from the spectrum

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`spectrum.plot`, `zm`, `fzm`

`Rgamma` *The Regularized Gamma Function (sfunc)*

Description

Computes the regularized Gamma function and its inverse. Both the lower and the upper regularized Gamma function are supported, and the Gamma value can be scaled to a base 10 logarithm.

Usage

```
Rgamma(a, x, lower=TRUE, log=FALSE)

Rgamma.inv(a, y, lower=TRUE, log=FALSE)
```

Arguments

`a`	a non-negative numeric vector, the parameter of the incomplete Gamma function
`x`	a non-negative numeric vector, the point at which the incomplete Gamma function is evaluated
`y`	a numeric vector, the values of the regularized Gamma function (or their base 10 logarithms if `log=TRUE`)
`lower`	if `TRUE`, computes the lower regularized Gamma function (default). Otherwise, computes the upper regularized Gamma function.
`log`	if `TRUE`, the Gamma values are base 10 logarithms (default: `FALSE`)

Details

The regularized Gamma functions scale the corresponding incomplete Gamma functions to the interval $[0, 1]$, by dividing through $\Gamma(a)$. Thus, the lower regularized Gamma function is given by

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)}$$

and the upper regularized Gamma function is given by

$$Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

Value

`Rgamma` returns the (lower or upper) regularized Gamma function with parameter `a` evaluated at point `x`.

`Rgamma.inv` returns the point `x` at which the (lower or upper) regularized Gamma function with parameter `a` evaluates to `y`.

See Also

`Cgamma`, `Igamma`, `Cbeta`, `Ibeta`, `Rbeta`

Examples

```
## P(X >= k) for Poisson distribution with mean alpha
alpha <- 5
k <- 10
Rgamma(k, alpha) # == ppois(k-1, alpha, lower=FALSE)
```

---

spectrum.plot *Comparative Plot of Word Frequency Spectra (lexstats)*

Description

Comparative plot of up to five word frequency spectra (see Baayen, 2001), either as a side-by-side barplot or as points and lines on a logarithmic scale.

Usage

```
spectrum.plot(spc, m.max=Inf, log=FALSE, y.min=100, y.max=0,
              xlab="m", ylab="V_m / E[V_m]",
              legend=NULL,
              pch=c(1, 3, 15, 2, 20),
              lwd=1,
              lty=c("solid", "dashed", "dotdash", "dotted", "twodash"),
              col=if (log) c("black") else c("black", "grey50", ...))
```

Arguments

`spc`	a list containing up to five frequency spectrum vectors. Such spectrum vectors can be read in from a file in `lexstats` format with `read.spectrum` or generated by a ZM or fZM model with the `EVm` method.
`m.max`	number of frequency ranks to be shown in plot. If unspecified, it is determined by the shortest spectrum vector in `spc`.
`log`	if `TRUE`, display frequency spectra as points and lines on a logarithmic scale. If `FALSE`, display spectra as side-by-side barplot on a linear scale (default). The latter is only useful when `m.max` is comparatively small.
`y.min, y.max`	range of y-axis. `y.max` is automatically computed to fit the data in `spc`. `y.min` is only used when `log=TRUE` and defaults to 100.
`legend`	a vector of character strings or expressions specifying the labels to be shown in a legend box. If `legend` is missing, no legend box will be displayed.
`xlab, ylab`	character strings giving labels for the x-axis and y-axis
`pch, lwd, lty`	vectors of plot symbols, line widths, and line types (only used if `log=TRUE`. Values are recycled if necessary. See the `par` manpage for possible ways of specifying these attributes.
`col`	a vector of colours for the lines (`log=TRUE`) or bars (`log=FALSE`) in the plot. Values are recycled if necessary. Colours are specified as described in the `par` manpage.

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`read.spectrum`, `zm`, `fzm`, `EVm`

`ucs.library` *Load UCS/R Modules*

Description

Since the UCS/R functions are imported into the global namespace, they are collected in various modules that can be loaded separately on demand. `ucs.library` loads a specified module. When called without arguments, it prints a listing of available modules.

Usage

```
ucs.library(name, all=FALSE)
```

Arguments

`name`	a character string giving the name of a *single* UCS/R module to be loaded. If omitted, a list of all available modules is displayed (see below).
`all`	if `TRUE`, all available modules are loaded

Details

Unlike the `library` and `package` functions, `ucs.library(module)` will read in the requested module even if it has already been loaded.

Value

Calling the `ucs.library` function without arguments returns a list of all available UCS/R modules as an object of class `"UCSLibList"`, which prints as a nicely formatted listing including one-line description. Use `names(ucs.library())` to obtain a plain vector of module names.

See Also

`UCS` for an overview of the UCS/R modules

Examples

```
print(ucs.library())  # list of available modules

ucs.library("base")   # load and manage UCS data sets
ucs.library("plots")  # evaluation graphs

ucs.library(all=TRUE) # load all modules
```

`ucs.par` *Graphics Parameters for Evaluation Graphs (plots)*

Description

Set default graphics parameters for the `evaluation.plot` function, similar to `par` for general graphics parameters. The current parameter values are queried by giving their names as character strings. The values can be set by specifying them as arguments in `name=value` form, or by passing a single list of named values.

Usage

```
ucs.par(...)

.ucs.PAR
```

Arguments

`...`	either character strings (or vectors) specifying the names of parameters to be queried, or parameters to be set in `named=value` form, or a single list of named values. Valid parameter names are described below.

Details

The current default parameters are stored in the global variable `.ucs.PAR`. They can be queried by giving their names as one or more character vectors to `ucs.par`. `ucs.par()` (no arguments) returns all UCS graphics parameters.

Parameters are set by specifying their names and the new values as `name=value` pairs. Such a list can also be passed as a single argument to `ucs.par`, which is typically used to restore previous parameter values (that have been saved in a list variable).

Value

When parameters are set, their former values are returned in an invisible named list. Such a list can be passed as a single argument to `ucs.par` to restore the parameter values.

When a single parameter is queried, its value is returned directly. When two or more parameters are queried, the result is a named list.

Note the inconsistency, which is the same as for `par`: setting one parameter returns a list, but querying one parameter returns a vector (or a scalar, i.e. a vector of length 1).

UCS Graphics Parameters

`col` A character or integer vector specifying line colours forup to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

`lty` A character or integer vector specifying line styles for up to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

`lwd` A numeric vector specifying line widths for up to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

`bw.col` The line colours used in B/W mode (see the `evaluation.plot` manpage for details).

`bw.lty` The line styles used in B/W mode.

`bw.lwd` The line widths in B/W mode.

`n.first` The smallest n-best list to be evaluated (default: 100). Shorter n-best lists typically lead to highly unstable evaluation graphs. It may be necessary to set `n.first` to a higher value for evaluation based on random samples (cf. `evaluation.plot`).

`n.step` The step width for n-best lists in evaluation graphs (default: 1). The default setting evaluates all possible n-best lists. Higher values speed up computation, make graphs look less jagged, and reduce the size of PostScript files. A useful range is 5 . . . 20, depending on the size of the data set file.

`test.step` Step width for n-best lists where significance tests for result differences are applied, as a multiple of `n.step` (default: 10). Since these tests are time-consuming and significant differences are indicated by fairly large symbols in the plot, values below 5 are rarely useful.

`cex` A character expansion factor for labels, annotations, and symbols in evaluation plots (see `par` for details).

`lex` This parameter can be used to increase the line widths of evaluation graphs and some decorations. Not that `lex` is not an expansion factor, but is simply *added* to all line widths in the plot.

`do.file` If `FALSE`, `evaluation.file` will not generate PostScript files, which is useful while testing and fine-tuning plots (default: `TRUE`).

See Also

`evaluation.plot`, `evaluation.file`, `par`

Examples

```
print(names(ucs.par()))         # list available parameters

ucs.par("col", "lty", "lwd")    # the default line styles
ucs.par(c("col", "lty", "lwd")) # works as well

## temporary changes to graphics paramters:
par.save <- ucs.par(n.first=200, n.step=5)
## plots use the modified parameters here
ucs.par(par.save)               # restore previous values

ucs.library("plots")            # reload module for factory defaults
```

`UCS` *Introduction to UCS/R*

Description

UCS/R consists of a set of R libraries related to the visualisation of cooccurrence data and the evaluation of association measures. The current functionaliy includes: evaluation graphs for association measures (in terms of precision and recall), measures for inter-annotator agreement, and two population models for word frequency distributions.

Usage

```
source("/path/to/UCS/System/R/lib/ucs.R")
ucs.library()
```

Details

UCS/R is initialised by `source`ing the file 'ucs.R' in the 'lib/' subdirectory of the UCS/R directory tree. This will make the UCS/R documentation available in the R process and provide the `ucs.library` command, which is used to load individual UCS/R modules. Enter `ucs.library()` now to display a list of available modules (see the `ucs.library` manpage for details).

Currently, the following modules are available. The listing below also indicates the most important manpages for each module. Throughout the documentation, it is assumed that you are familiar with the UCS/Perl naming conventions and data set file format.

- `sfunc`: **Special Mathematical Functions**
  Convenience interfaces to the Gamma function (`Cgamma`), the incomplete (and regularized) Gamma function and its inverse (`Igamma`, `Rgamma`), the Beta function (`Cbeta`), the incomplete (and regularized) Beta function and its inverse (`Ibeta`, `Rbeta`), and binomial confidence intervals (`binom.conf.interval`).
  All these functions are computed from the `pgamma` and `pbeta` distributions (and the corresponding quantile functions) in the standard library of R.
- `base`: **Basic Functions for Loading and Managing UCS data sets**
  This module provides functions for loading UCS data set files (`read.ds.gz`), listing annotated association measures (`ds.find.am`, `am.key2var`), ranking by association scores (`order.by.am`, `add.ranks`), and computing precision/recall tables for the evaluation of association measures (`precision.recall`).
  The module also includes a listing of all built-in association measures in the UCS/Perl system, including add-on packages (`builtin.ams`).
- `plots`: **Evaluation Graphs for Association Measures**
  This module plots precision-, recall-, and precision-by-recall graphs for the empirical evaluation of association measures (all combined in a single function, `evaluation.plot`). The graphs are highly configurable, either locally in each function call or by setting global default (`ucs.par`). The `evaluation.plot` function supports confidence intervals, significance tests for result differences, and evaluation based on random samples (see Evert, 2004, Ch. 5).
- `iaa`: **Measures of Inter-Annotator Agreement**
  Computes Cohen's kappa statistic with standard deviation (Fleiss, Cohen & Everitt, 1969) or confidence interval for proportion of truee agreement (Krenn, Evert & Zinsmeister, 2004) from a 2×2 contingency table (see `iaa.kappa` and `iaa.pta`)
- `lexstats`: **Interface to the `lexstats` Software**
  These are the beginnings of a rudimentary interface to the `lexstats` software for the analysis of word frequency distributions (Baayen, 2001). Currently, only the `read.spectrum` and `spectrum.plot` functions are useful.
- `zm`: **The Zipf-Mandelbrot (ZM) Population Model**
  This module implements a simple population model for word frequency distributions (Baayen, 2001) based on the Zipf-Mandelbrot law. See (Evert, 2004a) for details. Relevant help pages are `zm`, `EV`, `EVm`, `write.lexstats`, and `lnre.goodness.of.fit`.
- `fzm`: **The Finite Zipf-Mandelbrot (fZM) Population Model**
  This module implements the finite Zipf-Mandelbrot model, an extension of the ZM model (Evert, 2004a). Relevant help pages are `fzm`, `EV`, `EVm`, `write.lexstats`, and `lnre.goodness.of.fit`.

The command `help(package=UCS)` will give you a full index of available UCS/R help pages. Use `help.search()` for full-text search.

Note

The correct `source` path for the file 'ucs.R' can be set automatically with the UCS/R tool `ucs-config`. Simply insert the statement

```
source("ucs.R")
```

on a separate line in your R script file (say, 'my-script.R') and run the shell command

```
ucs-config my-script.R
```

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

Fleiss, Joseph L.; Cohen, Jacob; Everitt, B. S. (1969). Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin*, **72**(5), 323–327.

Krenn, Brigitte; Evert, Stefan; Zinsmeister, Heike (2004). Determining intercoder agreement for a collocation identification task. In preparation.

See Also

`ucs.library`, the UCS/R tutorial ('tutorial.R' in the 'script/' subdirectory) and the UCS/Perl documentation.

`write.lexstats` *Write Data Files for Goodness-of-Fit Evaluation of LNRE Model (zm, fzm)*

Description

Creates three data files in `lexstats` format, which can be used to evaluate the goodness-of-fit of a LNRE model with a multivariate chi-squared test (Baayen, 2001, Sec. 3.3), using the `lnreChi2` program (Baayen, 2001).

Usage

```
write.lexstats(model, file)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model. The object must include observed word frequency data (in components `N`, `V`, and `spc`), usually because the model parameters have been estimated from the observed frequency spectrum.
`file`	a character string giving the basename of the files that will be created

Details

This functions creates files in `lexstats` format with the extensions `.spc`, `.sp2`, and `.ev2`, which are required by the `lnreChi2` tool (Baayen, 2001, 270).

In addition, the basename `file` is extended with the string `"_bZM"` (for a ZM model) or `"_bfZM"` (for a fZM model), so that the `lnreChi2` tool can correctly identify the number of degrees of freedom (reduced by two estimated parameters for the ZM model, and three estimated parameters for the fZM model).

Value

The full basename of the created files (obtained by adding a model-specific suffix to the `file` parameter).

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`, `EV`, `EVm`

`zm` *The Zipf-Mandelbrot LNRE Model (zm)*

Description

Object constructor for a Zipf-Mandelbrot (ZM) LNRE model with parameters α and C (see Evert, 2004a for details). Either the parameters are specified explicitly, or one or both of them can be estimated from an observed frequency spectrum.

Usage

```
zm(alpha, C)

zm(alpha, N, V)

zm(N, V, spc, m.max=15, stepmax=10, debug=FALSE)
```

Arguments

`alpha`	a number in the range $(0, 1)$, the shape parameter α of the ZM model. `alpha` can automatically be estimated from `N`, `V`, and `spc`.
`C`	a positive number, the parameter C of the ZM model. `C` can automatically be estimated from `N` and `V`.
`N`	the sample size, i.e. number of observed tokens
`V`	the vocabulary size, i.e. the number of observed types
`spc`	a vector of non-negative integers representing the class sizes V_m of the observed frequency spectrum. The vector is usually read from a file in `lexstats` format with the `read.spectrum` function.

`m.max`	the number of ranks from `spc` that will be used to estimate the α parameter
`stepmax`	maximal step size of the `nlm` function used for parameter estimation. It should not be necessary to change the default value.
`debug`	if `TRUE`, print debugging information during the parameter estimation process. This feature can be useful to find out why parameter estimation fails.

Details

The ZM model with parameters $\alpha \in (0, 1)$ and $C > 0$ is defined by the type density function

$$g(\pi) := C \cdot \pi^{-\alpha-1}$$

for $0 \leq \pi \leq B$, where the upper bound B is determined from C by the normalisation condition

$$\int_0^\infty \pi \cdot g(\pi) d\pi = 1$$

The parameter α is estimated by nonlinear minimisation (`nlm`) of a multinomial chi-squared statistic for the observed against the expected frequency spectrum. Note that this is different from the multivariate chi-squared test used to measure the goodness-of-fit of the final model (Baayen, 2001, Sec. 3.3).

Value

An object of class `"zm"` with the following components:

`alpha`	value of the α parameter
`B`	value of the upper bound B (a normalisation device)
`C`	value of the C parameter
`N`	number of observed tokens (if specified)
`V`	number of observed types (if specified)
`spc`	observed frequency spectrum (if specified)

This object `print`s a short summary, including a comparison of the first ranks of the observed and expected frequency spectrum (if available).

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

See Also

`fzm`, `EV`, `EVm`, `write.lexstats`, `read.spectrum`, and `spectrum.plot`

Index