Package ‘kanjistat’

June 4, 2024

Type Package

Title A Statistical Framework for the Analysis of Japanese Kanji Characters

Version 0.14.1

Date 2024-05-30

Maintainer Dominic Schuhmacher <dominic.schuhmacher@mathematik.uni-goettingen.de>

Description Various tools and data sets that support the study of kanji, including their morphology, decomposition and concepts of distance and similarity between them.

URL https://dschuhmacher.github.io/kanjistat/

BugReports https://github.com/dschuhmacher/kanjistat/issues

Depends R (>= 4.1)

Imports methods, graphics, grDevices, utils, crayon, dendextend, gsubfn, Matrix, png, purrr, RANN, rlang, ROI, sysfonts, showtext, stringi, stringr, transport (>= 0.15), xml2, lifecycle, Rcpp

Suggests dplyr, jsonlite, kanjistat.data, knitr, rmarkdown, ROI.plugin.glpk, systemfonts, testthat (>= 3.0.0), tibble, withr

Additional_repositories https://dschuhmacher.github.io/drat

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

VignetteBuilder knitr

Config/testthat/edition 3

LinkingTo Rcpp

NeedsCompilation yes

Author Dominic Schuhmacher [aut, cre] (<https://orcid.org/0000-0001-7079-6313>), Lennart Finke [aut] (<https://orcid.org/0009-0003-6908-314X>)

Repository CRAN

Date/Publication 2024-06-04 15:40:02 UTC

Contents

`cjk_escape` *Replace CJK characters in files by escape sequences*

Description

All CJK characters in the file(s) found at the specified path are substituted by their Unicode escape sequences (\u + 4 digit hex number or \U + 8 digit hex number where necessary).

Usage

```
cjk_escape(path, outdir = NULL, verbose = TRUE)
```

Arguments

`path` the path to a directory or a single file.

`outdir` the directory where the output files are written. Defaults to the subdirectory `out` of the directory in `path`. The output files have the same names as the originals.

`verbose` whether to print a message for each output file.

Details

If `path` is a directory, the replacement is performed for all files at that location (subdirectories are ignored). If `outdir` is the same as `path`, the original files are overwritten without warning.

If `path` is a file, the replacement is limited to this file. If `outdir` is the same as `dirname(path)`, the files are overwritten without warning.

Value

No return value, called for side effects.

`codepoint` *Convert between Unicode codepoint and kanji*

Description

Given codepoints `cp`, the function `codepointToKanji` transforms to UTF-8, which will typically show as the actual character the codepoints stands for. Vice versa, given (UTF-8 encoded) kanjis `kan`, the function `kanjiToCodepoint` transforms to unicode codepoints.

Usage

```
codepointToKanji(cp, concat = FALSE)

kanjiToCodepoint(kan, character = FALSE)
```

Arguments

`cp` a vector of character strings or objects of class `hexmode`, representing hexadecimal numbers.

`concat` logical. Shall the returned characters be concatenated?

`kan` a vector of kanjis (strings of length 1) or a single string of length >= 1 of kanjis.

`character` logical. Shall the returned codepoints be of class "character" or hexmode.

Value

For `codepointToKanji` a character vector of kanji. For `kanjiToCodepoint` a vector of hexadecimal numbers (class `hexmode`).

Examples

```
codepointToKanji(c("51b7", "6696", "71b1"))
kanjiToCodepoint("\u51b7\u6696\u71b1")
```

compare_neighborhoods *Compare distances of nearest kanji*

Description

List distances to nearest neighbors of a given kanji in terms of a reference distance (which is currently only the stroke edit distance) and compare with values in terms of another distance (currently only the component transport distance, a.k.a. kanji distance).

Usage

```
compare_neighborhoods(
  kan,
  refdist = "strokedit",
  refnn = 10,
  compdist = "kanjidist",
  compnn = 0,
  ...
)
```

Arguments

kan	a kanji (currently only as a single UTF-8 character).
refdist	the name of the reference distance (currently only "strokedit").
refnn	the number of nearest neighbors in terms of the reference distance.
compdist	a character vector. The name(s) of one or several other distances to compare with (currently only "kanjidist").
compnn	the number of nearest neighbors in terms of the other distance(s). If this is positive it is assumed that the suggested package kanjistat.data is available.
...	further parameters that are passed to `kanjidist()`.

Value

A matrix of distances with `refnn + compnn` columns named by the nearest neighbors of `kan` (first in terms of the reference distance, then the other distances) and `1 + length(compdist)` rows named by the type of distance.

Warning

[Experimental]
This is only a first draft of the function and its interface and details may change considerably in the future. As there is currently no precomputed kanjidist matrix, there is a huge difference in computation time between setting compnn = 0 (only kanji distances to the refnn nearest neighbors in terms of refdist have to be computed) and setting compnn to any value $> 0$ (kanji distances to all 2135 other Jouyou kanji have to be computed in order to determine the compnn nearest neighbors; depending on the system and parameter settings this can take (roughly) anywhere between 2 minutes and an hour).

Examples

```
# compare_neighborhoods("\u6674", refnn=5, compo_seg_depth=4, approx="pcweighted",
#                        compnn=0, minor_warnings=FALSE)
```

---

convert_kanji   *Convert between kanji formats*

Description

Accept any interpretable representation of kanji in terms of index numbers, UTF-8 character strings of length 1, UTF-8 codepoints or kanjivec objects and convert it to all or any of these formats.

Usage

```
convert_kanji(
  key,
  output = c("all", "index", "character", "hexmode", "kanjivec"),
  simplify = TRUE
)
```

Arguments

key	an atomic vector or list of kanji in any combination of formats.
output	a string describing the desired output.
simplify	logical. Whether to simplify the output to an atomic vector or keep the structure of the original vector. In either case it depends on output whether this is possible.

Details

Index numbers are in terms of the order in kbase. UTF-8 codepoints are usually of class "hexmode", but character strings starting with "0x" or "0X" are also accepted in the key.

For output = "kanjivec", the GitHub package kanjistat.data has to be available or an error is returned. For output = "all", component kanjivec is set to NA if kanjistat.data is not available.

Value

A vector of the same length as key. If `simplify` is `TRUE`, this is an atomic vector for output = "index", "character" or "hexmode", and a list for output = "kanjivec" or "all" a list. If `simplify` is `FALSE`, the original structure (atomic or list) kept whenever possible.

Examples

```
convert_kanji(as.hexmode("99ac"))
convert_kanji("0x99ac")  # same
convert_kanji(500, "character") == kbase$kanji[500]  # TRUE
```

`distdata` *Precomputed kanji distances*

Description

Precomputed kanji distances

Usage

```
dstrokedit

dyehli
```

Format

Symmetric sparse matrices containing distances between a key kanji, its ten nearest neighbors and possibly some other close kanji. For `dstrokedit`, these are the stroke edit distances according to Yencken and Baldwin (2008). For `dyehli`, these are the bag-of-radicals distances according to Yeh and Li (2002). Both are an instance of the S4 class `dsCMatrix` (symmetric sparse matrices in *column*-compressed format) with 2133 rows and 2133 columns.

All pre-2010 jouyou kanji that are also post-2010 jouyou kanji are included. The indices are those from `kbase`.

Source

Datasets from `https://lars.yencken.org/datasets`, made available under the Creative Commons Attribution 3.0 Unported licence.

Computed as part of *Yencken, Lars (2010) Orthographic support for passing the reading hurdle in Japanese. PhD Thesis, University of Melbourne, Melbourne, Australia.*

References

Yeh, Su-Ling and Li, Jing-Ling (2002). Role of structure and component in judgements of visual similarity of Chinese characters. *Journal of Experimental Psychology: Human Perception and Performance*, **28**(4), 933–947.

Yencken, Lars, & Baldwin, Timothy (2008). Measuring and predicting orthographic associations: Modelling the similarity of Japanese kanji. In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pp. 1041-1048.

Examples

```
# Find index for kanji \u90e8
bu_index <- match("\u90e8", kbase$kanji)

# Look up available stroke edit distances for \u90e8.
non_zero <- which(dstrokedit[bu_index,] != 0)
sed <- dstrokedit[non_zero, bu_index]
names(sed) <- kbase[non_zero,]$kanji
sort(sed)

# Look up available bag-of-radicals distances for \u90e8.
non_zero <- which(dyehli[bu_index,] != 0)
bord <- dyehli[non_zero, bu_index]
names(bord) <- kbase[non_zero,]$kanji
sort(bord)
```

`fivebetas` *A sample list of kanjivec objects*

Description

A sample list of kanjivec objects

Usage

```
fivebetas
```

Format

`fivebetas` is a list of five `kanjivec` objects representing the basic kanji \u90e8,\u969c,\u966a,\u90f5,\u9663 containing "beta" components, which come in fact from two different classical radicals:

- \u961c–>\u2ed6 on the left: mound, small village
- \u9091–>\u2ecf on the right: large village

Source

The list has been generated with the function kanjivec with parameter flatten="intelligent" from the corresponding files in the KanjiVG database by Ulrich Apel (https://kanjivg.tagaini.net/).

Examples

```
oldpar <- par(mfrow = c(1,5), mai = rep(0,4))
invisible( lapply(fivebetas, plot, seg_depth = 2) )
par(oldpar)
```

fivetrees *Sample lists of kanjimat objects*

Description

Sample lists of kanjimat objects

Usage

```
fivetrees1

fivetrees2

fivetrees3
```

Format

fivetrees1, fivetrees2 and fivetrees3 are lists of five kanjimat objects each, representing the same five basic kanji \u6821,\u6728,\u4f11,\u6797,\u76f8, containing each a tree component. Their matrices are antialiased 64 x 64 pixel representations of the kanji. The size is chosen as a compromise between aesthetics and memory/computational cost, such as for kmatdist.

All of them are in handwriting style fonts. fivetrees1 is in a Kyoukasho font (schoolbook style), fivetrees2 is in a Kaisho font (regular script calligraphy font), fivetrees3 is in a Gyousho font (semi-cursive calligraphy font).

An object of class list of length 5.

An object of class list of length 5.

An object of class list of length 5.

Source

The list has been generated with the function kanjimat using the Mac OS pre-installed YuKyokasho font (fivetrees1), as well as the freely available fonts nagayama_kai by Norio Nagayama and Kouzan-BrushFontGyousyo by Aoyagi Kouzan.

Examples

```
oldpar <- par(mfrow = c(3,5))
invisible( lapply(fivetrees1, plot) )
invisible( lapply(fivetrees2, plot) )
invisible( lapply(fivetrees3, plot) )
par(oldpar)
```

get_strokes *Get the strokes of a kanjivec object*

Description

The strokes are the leaves of the kanjivec stroketree. They consist of a two-column matrix giving a discretized path for the stroke in the unit square $[0,1]^2$ with further attributes.

Usage

```
get_strokes(kvec, which = 1:kvec$nstrokes, simplify = TRUE)
```

Arguments

kvec	an object of class kanjivec
which	a numeric vector specifying the numbers of the strokes that are to be returned. Defaults to all strokes.
simplify	logical. Shall only the stroke be returned if which has length 1?

Value

Usually a list of strokes with attributes. Regardless of whether which is ordered or contains duplicates, the returned list will always contain the strokes in their natural order without duplicates. If which has length 1 and simplified = TRUE, the list is avoided, and only the single stroke is returned.

See Also

get_strokes_compo

Examples

```
kanji <- fivebetas[[5]]
get_strokes(kanji, c(3,10))    # the two long vertical strokes in \u9663
```

get_strokes_compo *Get the strokes of a specific component of a kanjivec object*

Description

The strokes are the leaves of the kanjivec `stroketree`. They consist of a two-column matrix giving a discretized path for the stroke in the unit square $[0, 1]^2$ with further attributes.

Usage

```
get_strokes_compo(kvec, which = c(1, 1))
```

Arguments

`kvec` an object of class `kanjivec`

`which` a vector of length 2 specifing the index of the component, i.e. the component used is `pluck(kvec$components, !!!which)`. The default `c(1,1)` refers to the root component (full kanji), so all strokes are returned.

Value

A list of strokes with attributes.

See Also

`get_strokes`

Examples

```
kanji <- fivebetas[[5]]
# get the three strokes of the component\u2ed6 in \u9663
rad <- get_strokes_compo(kanji, c(2,1))
plot(0.5, 0.5, xlim=c(0,1), ylim=c(0,1), type="n", asp=1, xaxs="i", yaxs="i", xlab="", ylab="")
invisible(lapply(rad, lines, lwd=4))
```

kanjidata *Data on kanji*

Description

The tibbles kbase and kmorph provide basic and morphologic information, respectively, for all kanji contained in the KANJIDIC2 file (see below)

Usage

```
kbase

kmorph
```

Format

kbase is a tibble with 13,108 rows and 13 variables:

kanji the kanji

unicode the Unicode codepoint

strokes the number of strokes

class one of four classes: "kyouiku", "jouyou", "jinmeiyou" or "hyougai"

grade a number from 1-11, basically a finer version of class, same as in KANJIDIC2, except that we assgined an 11 for all hyougaiji (rather than an NA value)

kanken at what level the kanji appears in the Nihon Kanji Nouryoku Kentei (Kanken)

jlpt at what level the kanji appears in the Japanese Language Proficiency Test (Nihongou Nouryoku Shiken)

wanikani at what level the kanji is learned on the kanji learning website Wanikani

frank the frequency rank (1 = most frequent) "based on several averages (Wikipedia, novels, newspapers, ...)"

frank_news the frequency rank (1 = most frequent) based on news paper data (2501 most frequent kanji over four years in the Mainichi Shimbun)

read_on, read_kun a single ON reading in katakana

read_kun a single kun reading in hiragana

mean a single English meaning of the kanji

kmorph is a tibble with 13,108 rows and 15 variables:

kanji the kanji

strokes the number of strokes

radical the traditional (Kangxi) radical used for indexing kanji (one of 214)

radvar the variant of the radical if it is different, otherwise `NA`

nelson_c the Nelson radical if it differs from the traditional one, otherwise `NA`

idc ideographic description character (plus sometimes a number or a letter) describing the shape of the kanji

components visible components of the kanji; originally from KRADFILE

skip the kanji's SKIP code

mean a single English meaning of the kanji (same as in kbase)

Details

The single ON and kun readings and the single meaning are for easy identification of the more difficult kanji. They are the first entry in the KANJIDIC2 file which may not always be the most important one. For full readings/meanings use the function `lookup` or consult a dictionary.

Source

Most of the data is directly from the KANJIDIC2 file. https://www.edrdg.org/wiki/index.php/KANJIDIC_Project
Variables `jlpt`, `frank`, `idc`, `components` were taken from the Kanjium data base https://github.com/mifunetoshiro/kanjium
Variable `components` is originally from RADKFILE/KRADFILE. https://www.edrdg.org/)

The use of this data is covered in each case by a Creative Commons BY-SA 4.0 License. See the package's LICENSE file for details and copyright holders.

Variable "class" is derived from "grade".
Variable "kanken" was compiled based on the Wikipedia description of the test levels (as of September 2022).

`kanjidist` *Compute distance between two kanjivec objects based on hierarchical optimal transport*

Description

The kanji distance is based on matching hierarchical component structures in a nesting-free way across all levels. The cost for matching individual components is a cost for registering the components (i.e. alligning there position, scale and aspect ratio) plus the (relative unbalanced) Wasserstein distance between the registered components.

Usage

```
kanjidist(
  k1,
  k2,
  compo_seg_depth1 = 3,
  compo_seg_depth2 = 3,
  p = 1,
  C = 0.2,
  approx = c("grid", "pc", "pcweighted"),
  type = c("rtt", "unbalanced", "balanced"),
  size = 48,
  lwd = 2.5,
  density = 30,
  verbose = FALSE,
  minor_warnings = TRUE
)
```

Arguments

`k1, k2`	two objects of type `kanjivec`.

`compo_seg_depth1`, `compo_seg_depth2`
two integers ≥ 1. Specifies for each kanji the deepest level included for component matching. If 1, only the kanji itself is used.

`p` the order of the Wasserstein distance used for matching components. All distances and the penalty (if any) are taken to the `p`-th power (which is compensated by taking the `p`-th root after summation).

`C` the penalty for extra mass if `type` is `"rtt"` or `"unbalanced"`, i.e. we add `C^p` per unit of extra mass (before applying the `p`-th root).

`approx` what kind of approximation is used for matching components. If this is `"grid"`, a bitmap (raster image) is used, otherwise lines are approximated by more freely spaced points. For `"pc"` (point cloud) each point has the same weight and points are placed in a (more or less) equidistant way. For `"pcweighted"` points are further apart along straight lines and around the center of the Bezier curves that describe the strokes. The weights of the points are then (more or less) proportional to the amount of ink (stroke length) they represent.

`type` the type of Wasserstein distance used for matching components based on the grid or point cloud approximation chosen. `"unbalanced"` means the weights (pixel values if `approx = "grid`) are interpreted as mass. The total masses in two components be very different. Extra mass can be disposed of at cost `C^p` per unit. `"rtt"` is computationally the same, but the final distance is divided by the maximum of the total ink (sum of weights) in each component to the 1/p. `"balanced"` means the weights are normalized so that both images have the same total mass 1. Everything has to be transported, i.e.\ disposal of mass is not allowed.

`size` side length of the bitmaps used for matching components (if `approx = "grid`).

`lwd` linewidth for drawing the components in these bitmaps (if `approx = "grid`).

`density` approximate number of discretization points per unit line length (if `approx != "grid`)

`verbose` logical. Whether to print detailed information on the cost for all pairs of components and the final matching.

`minor_warnings` logical. Should minor_warnings be given. If `FALSE`, the warnings about substantial distances between bitmaps/pointclouds standing for the same component and the use of a workaround due to missing strokes in component decompositions are suppressed. While these warnings indicate to same extent that things are not going exactly as planned, they are usually not of interest if a larger number of kanji distances is computed and obscure the visibility of more important warnings (if any).

Details

For the precise definition and details see the reference below. Parameter `C` corresponds to $b/2^{1/p}$ in the paper.

Value

The kanji distance, a non-negative number.

Warning

[Experimental]
The interface and details of this function will change in the future. Currently only a minimal set of parameters can be passed. The other parameters are fixed exactly as in the "prototype distance" (4.1) of the reference below for better or worse.
There is a certain tendency that exact matches of components are rather strongly favored (if the KanjiVG elements agree this can overrule the unbalanced Wasserstein distance) and the penalties for translation/scaling/distortion of components are somewhat mild.
The computation time is rather high (depending on the settings and kanji up to several seconds per kanji pair). This can be alleviated somewhat by keeping the `compo_seg_depth` parameters at 3 or lower and setting `size = 32` (which goes well with `lwd=1.8`).
Future versions will use a much faster line base optimal transport algorithm and further speed-ups.

References

Dominic Schuhmacher (2023).
Distance maps between Japanese kanji characters based on hierarchical optimal transport.
ArXiv, doi:10.48550/arXiv.2304.02493

See Also

`kanjidistmat`, `kmatdist`

Examples

```
if (requireNamespace("ROI.plugin.glpk")) {
  kanjidist(fivebetas[[4]], fivebetas[[5]])
  kanjidist(fivebetas[[4]], fivebetas[[5]], verbose=TRUE)
  # faster and similar:
  kanjidist(fivebetas[[4]], fivebetas[[5]], compo_seg_depth1=2, compo_seg_depth2=2,
            size=32, lwd=1.8, verbose=TRUE)
  # slower and similar:
  kanjidist(fivebetas[[4]], fivebetas[[5]], size=64, lwd=3.2, verbose=TRUE)
}
```

`kanjidistmat` *Compute distance matrix based on hierarchical optimal transport for lists of kanjivec objects*

Description

Individual distances are based on `kanjidist`.

Usage

```
kanjidistmat(
  klist,
  klist2 = NULL,
  compo_seg_depth = 3,
  p = 1,
  C = 0.2,
  approx = c("grid", "pc", "pcweighted"),
  type = c("rtt", "unbalanced", "balanced"),
  size = 48,
  lwd = 2.5,
  density = 30,
  verbose = FALSE,
  minor_warnings = FALSE
)
```

Arguments

`klist` a list of `kanjimat` objects.

`klist2` an optional second list of `kanjimat` objects.

`compo_seg_depth`
integer ≥ 1. Specifies for all kanji the deepest level included for component matching. If 1, only the kanji itself is used.

`p`, `C`, `type`, `approx`, `size`, `lwd`, `density`, `verbose`, `minor_warnings`
the same as for the function `kanjidist`, with the sole difference that `minor_warnings` defaults to `FALSE` here.

Value

A matrix of dimension `length(klist)` x `length(klist2)` having as its (i, j)-th entry the distance between `klist[[i]]` and `klist2[[j]]`. If `klist2` is not provided it is assumed to be equal to `klist`, but computation is more efficient as only the upper triangular part is computed and then symmetrized with diagonal zero.

Warning

[Experimental]
The same precautions apply as for `kanjidist`.

See Also

`kanjidist`, `kmatdistmat`

Examples

```
kanjidistmat(fivebetas)
```

kanjimat *Create kanjimat objects*

Description

Create a (list of) kanjimat object(s), i.e. bitmap representations of a kanji using a certain font-family and other typographical parameters.

Usage

```
kanjimat(
  kanji,
  family = NULL,
  size = NULL,
  margin = 0,
  antialias = TRUE,
  save = FALSE,
  overwrite = FALSE,
  simplify = TRUE,
  ...
)
```

Arguments

`kanji`	a (vector of) character string(s) containing kanji.
`family`	the font-family to be used. For details see vignette.
`size`	the sidelength of the (square) bitmap
`margin`	numeric. Extra margin around the character. Defaults to 0 which leaves a relatively slim margin. Positive values increase this margin, negative values decrease it (which usually cuts off part of the kanji).
`antialias`	logical. Shall antialiasing be performed?
`save`	logical or character. If FALSE return the (list of) kanjimat object(s). Otherwise save the result as an rds file in the working directory (as kmatsave.rds) or under the file path provided.
`overwrite`	logical. If FALSE return an error (before any computations are done) if the designated file path already exists. Otherwise an existing file is overwritten.
`simplify`	logical. Shall a single kanjimat object be returned (instead a list of one) if `kanji` is a single kanji?
`...`	futher arguments passed to png. This is for extensibility. The only argument that may currently be used is `type`. Trying to change sizes, units, colors or fonts by this argument results in an error or an undesirable output.

Value

A list of objects of class `kanjimat` or, if only one kanji was specified and `simplify` is `TRUE`, a single objects of class `kanjimat`. If `save = TRUE`, the same is (saved and) still returned invisibly.

Warning

If no font family is provided, the default **Chinese** font WenQuanYi Micro Hei that comes with the package showtext is used. This means that the characters will typically be recognizable, but quite often look odd as Japanese characters. We strongly advised that a Japanese font is used as detailed above.

Examples

```
res <- kanjimat(kanji="\u85e4", size = 128)
```

---

kanjivec *Create kanjivec objects from kanjivg data*

Description

Create a (list of) kanjivec object(s). Each object is a representation of the kanji as a tree of strokes based on .svg files from the KanjiVG database containing further, derived information.

Usage

```
kanjivec(
  kanji,
  database = NULL,
  flatten = "intelligent",
  bezier_discr = c("svgparser", "eqtimed", "eqspaced"),
  save = FALSE,
  overwrite = FALSE,
  simplify = TRUE
)
```

Arguments

`kanji`	a (vector of) character string(s) of one or several kanji.
`database`	the path to a local copy of (a subset of) the KanjiVG database. It is expected that the svg files reside at this exact location (not in a subdirectory). If `NULL`, an attempt is made to read the svg file(s) from the KanjiVG GitHub repository (after prompting for confirmation, which can be switched off via the option `ask_github`).
`flatten`	logical. Should nodes that are only-children be fused with their parents? Alternatively one of the strings "intelligent", "inner" or "leaves". Although the first is the default it is experimental and the precise meaning will change in the future; see details.

`bezier_discr` character. How to discretize the Bézier curves describing the strokes. If "svgparser" (the only option available prior to kanjistat 0.12.0), code from the non-CRAN package svgparser is used for discretizing at equal time steps. The new choices "eqtimed" and "eqspaced" discretize into fewer points (and allow for more customization underneath). The former creates discretization points at equal time steps, the latter at equal distance steps (to a good approximation).

`save` logical or character. If FALSE return the (list of) kanjivec object(s). Otherwise save the result as an rds file in the working directory (as kvecsave.rds) or under the file path provided.

`overwrite` logical. If FALSE return an error (before any computations are done) if the designated file path already exists. Otherwise an existing file is overwritten.

`simplify` logical. Shall a single kanjivec object be returned (instead a list of one) if `kanji` is a single kanji?

Details

A kanjivec object contains detailed information on the strokes of which an individual kanji is composed including their order, a segmentation into reasonable components ("radicals" in a more general sense of the word), classification of individual strokes, and both vector data and interpolated points to recreate the actual stroke in a Kyoukashou style font. For more information on the original data see http://kanjivg.tagaini.net/. That data is licenced under Creative Commons BY-SA 3.0 (see licence file of this package).

The original .svg files sometimes contain additional <g> elements that provide information about the current group of strokes rather than establishing a new subgroup of its own. This happens typically for information that establishes coherence with another part of the tree (by noting that the current subgroup is also part 2 of something else), but also for variant information. With the option `flatten = TRUE` the extra hierarchy level in the tree is avoided, while the original information in the KanjiVG file is kept. This is achieved by fusing only-children to their parents, giving the new node the name of the child and all its attributes, but prefixing `p.` to the attribute names of the parent (the parents' "names" attribute is discarded, but can be reconstructed from the parents' id). Removal of several hierarchies in sequence can lead to attribute names with multiple `p.` in front. Fusing to parents is suppressed if the parent is the root of the hierarchy (typically for one-stroke kanji), as this could lead to confusing results.

The options `flatten = "inner"` and `flatten = "leaves"` implement the above behavior only for the corresponding type of node (inner nodes or leaves). The option `flatten = "intelligent"` tries to find out in more sophisticated ways which flattening is desirable and which is not (it will flatten rather conservatively). Currently nodes without an element attribute that have only one child are flattened away (one example where this is reasonable is in kanji `kbase[187, ]`), as are nodes with an element attribute and only one child if this child is also an inner node and has the same element and part attribute as the parent, but both have no number (this would be problematic for any component-building code in the particular case of kanji `kbase[1111, ]`).

A `kanjivec` object has components

`char` the kanji (a single character)

`hex` its Unicode codepoint (integer of class `hexmode`)

`padhex` the Unicode codepoint padded with zeros to five digits (mode character) `family` the font on which the data is based. Currently only "schoolbook" (to be extended with "kaisho" at some point)

`nstrokes` the number of strokes in the kanji

`ncompos` a vector of the number of components at each depth of the tree

`nveins` the number of veins in the component structure

`strokedend` the decomposition tree of the kanji as an object of class `dendrogram`

`components` the component structure by segmentation depth (components can overlap) in terms of KanjiVG elements and their depth-first tree coordinates

`veins` the veins in the component structure. Each vein is represented as a two-column matrix that lists in its rows the indices of `components` (starting at the root, which in the component indexing is `c(1,1)`)

`stroketree` the decomposition tree of the kanji, a list containing the full information of the the KanjiVG file (except some top level attributes)

`stroketree` is a close representation of the KanjiVG svg file as list object with some serious nesting of sublists. The XML attributes become attributes of the list and its elements. The user will usually not have to look at or manipulate `stroketree` directly, but `strokedend` and `compents` are derived from it and other functions may process it further.

The main differences to the svg file are

1. the actual strokes are not only given as d-attributes describing Bézier curves, but but also as two-column matrices describing discretizations of these curves. These matrices are the actual contents of the innermost lists in `stroketree`, but are more conveniently accessed via the function `get_strokes`. Starting with version 0.13.0, there is also an additional attribute "beziermat", which describes the Bézier curves for the stroke in a 2 x (1+3n) matrix format. The first column is the start point, then each triplet of columns stands for control point 1, control point 2 and end point (=start point of the next Bézier curve if any).
2. The positions of the stroke numbers (for plotting) are saved as an attribute strokenum_coords to the entire stroke tree rather than a separate element.

`strokedend` is more easy to examine and work with due to various convenience functions for dendrograms in the packages `stats` and `dendextend`, including `str` and `plot.dendrogram`. The function `plot.kanjivec` with option `type = "dend"` is a wrapper for `plot.dendrogram` with reasonable presets for various options.

The label-attributes of the nodes of `strokedend` are taken from the element (for inner nodes) and type (for leaves) attributes of the .svg files. They consist of UTF-8 characters representing kanji parts and a combination of UTF-8 characters for representing strokes and may not represent well in all CJK fonts (see details of `plot.kanjivec`). If element and type are missing in the .svg file, the label assigned is the second part of the id-attribute, e.g. g5 or s9.

The `components` at a given level can be plotted, see `plot.kanjivec` with `type = "kanji"`. Both `components` and `veins` serve mainly for the computation of kanji distances.

Value

A list of objects of class `kanjivec` or, if only one kanji was specified and `simplify` is `TRUE`, a single objects of class `kanjivec`. If `save = TRUE`, the same is (saved and) still returned invisibly.

See Also

plot.kanjivec, str.kanjivec

Examples

```
if (interactive()) {
  # Try to load the svg file for the kanji from GitHub.
  res <- kanjivec("\u85e4", database=NULL)
  str(res)
}

fivebetas  # sample kanjivec data
str(fivebetas[[1]])
```

---

kmatdist *Compute the unbalanced or balanced Wasserstein distance between two kanjimat objects*

Description

This gives the dissimilarity of pixel-images of the kanji based on how far mass (or "ink") has to be transported to transform one image into the other.

Usage

```
kmatdist(
  k1,
  k2,
  p = 1,
  C = 0.2,
  type = c("unbalanced", "balanced"),
  output = c("dist", "all")
)
```

Arguments

k1, k2	two objects of type kanjimat.
p	the order of the Wasserstein distance. All distances and a potential penalty are taken to the p-th power (which is compensated by taking the p-th root after summation).
C	the penalty for extra mass if type="unbalanced", i.e. we add C^p per unit of extra mass (before applying the p-th root).
type	the type of Wasserstein metric. "unbalanced" means the pixel values in the two images are interpreted as mass. The total masses can be very different. Extra mass can be disposed of at cost C^p per unit. "balanced" means the pixel values are normalized so that both images have the same total mass 1. Everything has to be transported, i.e. disposal of mass is not allowed.

output the requested output. See return value below.

Value

If output = "dist", a single non-negative number: the unbalanced or balanced Wasserstein distance between the kanji. If output = "all" a list with detailed information on the transport plan and the disposal of pixel mass. See unbalanced for details.

See Also

kmatdistmat, kanjidist

Examples

```
res <- kmatdist(fivetrees1[[1]], fivetrees1[[5]], p=1, C=0.1, output="all")
plot(res, what="plan", angle=20, lwd=1.5)
plot(res, what="trans")
plot(res, what="extra")
plot(res, what="inplace")
```

---

kmatdistmat *Compute distance matrix for lists of kanjimat objects*

Description

Apply kmatdist to every pair of kanjimat objects to compute the unbalanced or balanced Wasserstein distance.

Usage

```
kmatdistmat(
  klist,
  klist2 = NULL,
  p = 1,
  C = 0.2,
  type = c("unbalanced", "balanced")
)
```

Arguments

klist	a list of kanjimat objects.
klist2	an optional second list of kanjimat objects.
p, C, type	the same as for the function kmatdist.

Value

A matrix of dimension length(klist) x length(klist2) having as its (i, j)-th entry the distance between klist[[i]] and klist2[[j]]. If klist2 is not provided it is assumed to be equal to klist, but the computation is more efficient as only the upper triangular part is computed and then symmetrized with diagonal zero.

See Also

kmatdist, kanjidistmat

Examples

```
kmatdistmat(fivetrees1)
kmatdistmat(fivetrees1, fivetrees1)  # same result but slower
kmatdistmat(fivetrees1, fivetrees2)  # note the smaller values on the diagonal
```

---

kreadmean *Kanji readings and meanings*

Description

Data set of all kanji readings and meanings from the KANJIDIC2 dataset in an R list format. For convenient access to this data use function lookup.

Usage

```
kreadmean
```

Format

An object of class list of length 13108.

Source

KANJIDIC2 file by Jim Breen and The Electronic Dictionary Research and Development Group (EDRDG)
https://www.edrdg.org/wiki/index.php/KANJIDIC_Project
The use of this data is covered by the Creative Commons BY-SA 4.0 License.

`lookup` *Look up kanji*

Description

Return readings and meanings or information from `kbase` or `kmorph`.

Usage

```
lookup(kanji, what = c("readmean", "basic", "morphologic"))
```

Arguments

`kanji`	a (vector of) character strings containing kanji.
`what`	the sort of information to display.

Details

This is a very basic interface for a quick lookup information based on exact knowledge of the kanji (provided by a Japanese input method or its UTF-8 code). Most of the information is based on the KANJIDIC2 file by EDRDG (see thank you page) Please use one of the many excellent online kanji dictionaries (see e.g.) more sophisticated lookup methods and more detailed results.

Value

If `what` is "readmean" the information is output with cat and there is no return value (invisible NULL) In the other cases the appropriate subsets of the tables kbase and kmorph are returned

Author(s)

Dominic Schuhmacher <`schuhmacher@math.uni-goettingen.de`>

Examples

```
lookup(c("\u6674", "\u66c7", "\u96e8"))
lookup("\u6674\u66c7\u96e8")   # same
```

`options` *Kanjistat Options*

Description

Set or examine global kanjistat options.

Usage

```
kanjistat_options(...)

get_kanjistat_option(x)
```

Arguments

`...`	any number of options specified as `name = value`
`x`	name of an option given as character string.

Value

`kanjistat_options` returns the list of all set options if there is no function argument. Otherwise it returns list of *all* old options. `get_kanjistat_option` returns the current value set for option `x` or NULL if the option is not set.

`plot.kanjimat` *Plot kanjimat object*

Description

Plot kanjimat object

Usage

```
## S3 method for class 'kanjimat'
plot(
  x,
  mode = c("dark", "light"),
  col = gray(seq(0, 1, length.out = 256)),
  ...
)
```

Arguments

`x` object of class kanjimat.

`mode` character string. If "dark" the original grayscale values are used, if "light" they are inverted. With the default grayscale color scheme the kanji is plotted white-on-black for "dark" and black-on-white for "light".

`col` a vector of colors. Typically 256 values are enough to keep the full information of an (antialiased) kanjimat object.

`...` further parameters passed to `image`.

Value

No return value, called for side effects.

`plot.kanjivec` *Plot kanjivec objects*

Description

Plot kanjivec objects

Usage

```
## S3 method for class 'kanjivec'
plot(
  x,
  type = c("kanji", "dend"),
  seg_depth = 0,
  palette = "Dark 3",
  pal.extra = 0,
  numbers = FALSE,
  offset = c(0.025, 0),
  family = NULL,
  lwd = 8,
  ...
)
```

Arguments

`x` an object of class `kanjivec`

`type` either "kanji" or "dend". Whether to plot the actual kanji, coloring strokes according to levels of segmentation, or to plot a representation of the tree structure underlying this segmentation. Among the following named parameters, only `family` is for use with `type = "dend"`; all others are for `type = "dend"`.

`seg_depth` an integer. How many steps down the segmentation hierarchy we use different colors for different groups. If zero (the default), only one color is used that can be specified with `col` passed via `...` as usual `palette` a valid name of a hcl palette (one of `hcl.pals()`). Used for coloring the components if `seg_depth` is > 0.

`pal.extra` an integer. How many extra colors are picked in the specified palette. If this is 0 (the default), palette is used with as many colors as we have components. Since many hcl palettes run from dark to light colors, the last (few) components may be too light. Increasing pal.extra then makes the component colors somewhat more similar, but the last component darker.

`numbers` logical. Shall the stroke numbers be displayed.

`offset` the (x,y)-offset for the numbers relative to the positions from kanjivg saved in the kanjivec object. Either a vector of length 2 specifying some fixed offset for all numbers or a matrix of dimension kanjivec$nstrokes times 2.

`family` the font-family for labeling the nodes if `type = dend`. See details.

`lwd` the usual line width graphics parameter.

`...` further parameters passed to `lines` if `type = "kanji"` and to `plot.dendrogram` if `type = "dend"`.

Details

Setting up nice labels for the nodes if `type = "dend"` is not easy. For many font families it appears that some "kanji components" cannot be displayed in plots even with the help of package `showtext` and if the font contains glyphs for the corresponding codepoints that display correctly in text documents. This concerns in increasing severity of the problem Unicode blocks 2F00–2FDF (Kangxi Radicals), 2E80–2EFF (CJK Radicals Supplement) and 31C0–31EF (CJK Strokes). For the strokes it seems nearly impossible which is why leaves are simply annotated with the number of the strokes.

For the other it is up to the user to find a suitable font and pass it via the argument font family. The default `family = NULL` first tries to use `default_font` if this option has been set (via `kanjistat_options`) and otherwise uses `wqy-microhei`, the Chinese default font that comes with package `showtext` and cannot display any radicals from the supplement.

On a Mac the experience is that "hiragino_sans" works well. In addition there is the issue of font size which is currently not judiciously set and may be too large for some (especially on-screen) devices. The parameter `cex` (via `...`) fixes this.

Value

No return value, called for side effects.

Examples

```
kanji <- fivebetas[[2]]
plot(kanji, type = "kanji", seg_depth = 2)
plot(kanji, type = "dend")
  # gives a warning if get_kanjistat_option("default_font") is NULL
```

plotkanji *Plot kanji*

Description

Write kanji to a graphics device.

Usage

```
plotkanji(
  kanji,
  device = "default",
  family = NULL,
  factor = 10,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

`kanji`	a vector of class character specifying one or several kanji to be plotted.
`device`	the type of graphics device where the kanji is plotted. Defaults to the user's default type according to `getOption("device")`.
`family`	the font family or families used for writing the kanji. Make sure to add the font(s) first by using `font_add`; see details. If `family` is a vector of several font families they are matched to the characters in `kanji` (and possibly recycled).
`factor`	a maginification factor applied to the font size (typically 12 points).
`width, height`	the dimensions of the device.
`...`	further parameters passed to the function opening the device (such as a file name for devices that create a file).

Details

This function writes one or several kanji to a graphics device in an arbitrary font that has been registered, i.e., added to the database in package `sysfonts`. For the latter say `font_add` or `font_families` to verify what fonts are available.

For further information see *Working with Japanese fonts* in `vignette("kanjistat", package = "kanjistat")`. `plotkanji` uses the package `showtext` to write the kanji in a large font at the center of a new device of the specified type. specify `device = "current"` to write the kanji to the current device. It is now recommended to simply use `graphics::text` in combination with `showtext::showtext_auto` instead.

Value

No return value, called for side effects.

Warning

If no font family is provided, the default **Chinese** font WenQuanYi Micro Hei that comes with the package showtext is used. This means that the characters will typically be recognizable, but quite often look odd as Japanese characters. We strongly advised that a Japanese font is used as detailed above.

Examples

```
plotkanji("\u6edd")
plotkanji("\u72ac\u732b\u9b5a")
```

`pooled_similarity` *Precomputed kanji distances*

Description

Precomputed kanji distances

Usage

```
pooled_similarity
```

Format

A tibble containing kanji similarity judgments by 3 "native or native-like" speakers of Japanese. For each row, the pivot kanji was compared to a list of potential distractors. From the distractors, the subjects selected one character which they found particularly easy to confuse with the pivot. For the exact methodology, see the original study referenced below.

Source

Datasets from `https://lars.yencken.org/datasets`, made available under the Creative Commons Attribution 3.0 Unported licence.

Collected as part of *Yencken, Lars (2010) Orthographic support for passing the reading hurdle in Japanese. PhD Thesis, University of Melbourne, Melbourne, Australia.*

References

Yencken, Lars, & Baldwin, Timothy (2008). Measuring and predicting orthographic associations: Modelling the similarity of Japanese kanji. In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pp. 1041-1048.

Examples

```
# Get kanji characters that were found to be easily confused with \u5927.
pooled_similarity[pooled_similarity$selected == "\u5927", ]$pivot
```

print.kanjivec *Print basic information about a kanjivec object*

Description

Print basic information about a kanjivec object

Usage

```
## S3 method for class 'kanjivec'
print(x, dend = FALSE, ...)
```

Arguments

x	an object of class kanjivec.
dend	whether to print the structure of the strokedend component.
...	further parameters passed to print.default.

Value

No return value, called for side effects.

read_kanjidic2 *Read a KANJIDIC2 file*

Description

Perform basic validity checks and transform data to a standardized list or keep as an object of class xml_document (package xml2).

Usage

```
read_kanjidic2(fpath = NULL, output = c("list", "xml"))
```

Arguments

fpath	the path to a local KANJIDIC2 file. If NULL (the default) the most recent KANJIDIC2 file is downloaded from https://www.edrdg.org/kanjidic/kanjidic2.xml.gz after asking for confirmation.
output	one of "list" or "xml". The desired type of output.

Details

KANJIDIC2 contains detailed information on all of the 13108 kanji in three main Japanese standards (JIS X 0208, 0212 and 0213). The KANJIDIC files have been compiled and maintained by Jim Breen since 1991, with the help of various other people. The copyright is now held by the Electronic Dictionary Research and Development Group (EDRDG). The files are made available under the Creative Commons BY-SA 4.0 license. See `https://www.edrdg.org/wiki/index.php/KANJIDIC_Project` for details on the contents of the files and their license.

If `output = "xml"`, some minimal checks are performed (high level structure and total number of kanji).

If `output = "list"`, additional validity checks of the lower level structure are performed. Most are in accordance with the file's Document Type Definition (DTD). Some additional check concern some common patterns that are true about the current KANJIDIC2 file (as of December 2023) and seem unlikely to change in the near future. This includes that there is always at most one `rmgroup` entry in `reading_meaning`. Informative warnings are provided if any of these additional checks fail.

Value

If `output = "xml"`, the exact XML document obtained from xml2::read_xml. If `output = "list"`, a list of lists (the individual kanji), each with the following seven components.

- `literal`: a single UTF-8 character representing the kanji.
- `codepoint`: a named character vector giving the available codepoints in the unicode and jis standards.
- `radical`: a named numeric vector giving the radical number(s), in the range 1 to 214. The number named `classical` is as recorded in the *KangXi Zidian* (1716); if there is a number named `nelson_c`, the kanji was reclassified in Nelson's *Modern Reader's Japanese-English Character Dictionary* (1962/74).
- `misc`: a list with six components
  - `grade`: the kanji grade level. 1 through 6 indicates a kyouiku kanji and the grade in which the kanji is taught in Japanese primary school. 8 indicates one of the remaining jouyou kanji learned in junior high school, and 9 or 10 are jinmeiyou kanji. The remaining (hyougai) kanji have `NA` as their entry.
  - `stroke_count`: The stroke count of the kanji, including the radical. If more than one, the first is considered the accepted count, while subsequent ones are common miscounts.
  - `variant`: a named character vector giving either a cross-reference code to another kanji, usually regarded as a variant, or an alternative indexing code for the current kanji. The type of variant is given in the name.
  - `freq`: the frequency rank (1 = most frequent) based on newspaper data. `NA` if not among the 2500 most frequent.
  - `rad_name`: a character vector. For a kanji that is a radical itself, the name(s) of the radical (if there are any), otherwise of length 0.
  - `jlpt`: The Japanese Language Proficiency Test level according to the old four-level system that was in place before 2010. A value from 4 (most elementary) to 1 (most advanced).

- dic_number: a named character vector (possibly of length 0) giving the index numbers (for some kanji with letters attached) of the kanji in various dictionaries, textbooks and flashcard collections (specified by the name). For Morohashi's *Dai Kan-Wa Jiten*, the volume and page number is also provided in the format moro.VOL.PAGE.
- query_code: a named character vector giving the codes of the kanji in various query systems (specified by the name). For Halpern's SKIP code, possible misclassifications (if any) of the kanji are also noted in the format mis.skip.TYPE, where TYPE indicates the type of misclassification.
- reading_meaning: a (possibly empty) list containing zero or more rmgroup components creating groups of readings and meanings (in practice there is never more than one rmgroup currently) as well as a component nanori giving a character vector (possibly of length 0) of readings only associated with names. Each rmgroup is a list with entries:
  - reading: a (possibly empty) list of entries named from among pinyin, korean_r, korean_h, vietnam, ja_on and ja_kun, each containing a character vector of the corresponding readings
  - meaning: a (possibly empty) list of entries named with two-letter (ISO 639-1) language codes, each containing a character vector of the corresponding meanings.

See Also

kanjidata, kreadmean

Examples

```
if (interactive()) {
  read_kanjidic2("kanjidic2.xml")
}
```

---

samplekan *Sample kanji from a set*

Description

Sample kanji from a set

Usage

```
samplekan(
  set = c("kyouiku", "jouyou", "jinmeiyou", "kanjidic"),
  size = 1,
  replace = FALSE,
  prob = NULL
)
```

Arguments

`set`	a character string specifying the set of kanjis to sample from.
`size`	a positive number, the number of samples.
`replace`	logical. Sample with replacement?
`prob`	currently without effect.

Value

a vector of length `size` containing the individual characters

Examples

```
(sam <- samplekan(size = 10))
lookup(sam)
```

`sedist` *Compute the stroke edit distances between two sets of kanji*

Description

Variants of the stroke edit distance proposed by Yencken (2010). Each kanji is encoded as sequence of stroke types according to its stroke order, using the type attribute from the kanjiVG data. Then the edit distance (a.k.a.\ Levenshtein distance) between sequences is computed and divided by the maximum of the number of strokes

Usage

```
sedist(k1, k2, type = c("full", "before_slash", "first"))
```

Arguments

`k1, k2`	atomic vectors or lists of kanji in any format that can be treated by `convert_kanji()`
`type`	the type of stroke edit distance to compute. See details.

Details

The kanjiVG type attribute is a single string composed of a CJK strokes Unicode character, an optional latin letter providing further information and possibly a variant (another CJK strokes character with optional letter) separated by "/". If `type` is "full"‘ a match is only counted if two strings are exactly the same, "before_slash" ignores any slashes and what comes after them, "first" only considers the first character of each string (so the first CJK stroke character) when counting matches.

The stroke edit distance used by Yencken (2010) is obtained by setting type = "all" (the default), except that the underlying kanjiVG data has significantly changed since then. Comparing with the values in dstrokedit we get an agreement of 96.3 percent, whereas the other distances disagree by a small amount (usually 1-2 edit operations).

Value

A `length(k1)` x `length(k2)` matrix of stroke edit distances.

Warning

Requires kanjistat.data package.

References

Yencken, Lars (2010). Orthographic support for passing the reading hurdle in Japanese. PhD Thesis, University of Melbourne, Australia

Examples

```
ind1 <- 384
k1 <- convert_kanji(ind1, "character")
ind2 <- which(dstrokedit[ind1,] > 0)
# dstrokedit contains only the "closest" kanji
k2 <- convert_kanji(ind2, "character")
row_a <- dstrokedit[ind1, ind2]
if (requireNamespace("kanjistat.data", quietly = TRUE)) {
  row_b <- sedist(k1, k2)
  mat <- rbind(row_a, row_b)
  rownames(mat) = c(k1, k1)
  colnames(mat) = k2
  mat
}
```

`str.kanjivec` *Compactly display the structure of a kanjivec object*

Description

Compactly display the structure of a kanjivec object

Usage

```
## S3 method for class 'kanjivec'
str(object, ...)
```

Arguments

`object`	an object of class `kanjivec`.
`...`	further parameters passed to `str` for all but the `stroketree` component of `object`.

Value

No return value, called for side effects.

Index