Package 'immApex'

August 12, 2025

Title Tools for Adaptive Immune Receptor Sequence-Based Machine and Deep Learning

Version 1.3.7

Description A set of tools to for machine and deep learning in R from amino acid and nucleotide sequences focusing on adaptive immune receptors. The package includes pre-processing of sequences, unifying gene nomenclature usage, encoding sequences, and combining models. This package will serve as the basis of future immune receptor sequence functions/packages/models compatible with the scRepertoire ecosystem.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

biocViews Software, ImmunoOncology, SingleCell, Classification, Annotation, Sequencing, MotifAnnotation

Depends R (>= 4.3.0)

Imports hash, httr, Matrix, matrixStats, methods, Rcpp, rvest, SingleCellExperiment, stats, stringr, utils

Suggests BiocStyle, dplyr, ggraph, ggplot2, igraph, knitr, markdown, Peptides, randomForest, rmarkdown, scRepertoire, spelling, testthat, tidygraph, viridis

SystemRequirements Python (via basilisk)

LinkingTo Rcpp

VignetteBuilder knitr

Language en-US

URL https://github.com/BorchLab/immApex/

BugReports https://github.com/BorchLab/immApex/issues

git_url https://git.bioconductor.org/packages/immApex

git_branch devel

git_last_commit b751eba

git_last_commit_date 2025-08-11

Repository Bioconductor 3.22

Date/Publication 2025-08-11

Author Nick Borcherding [aut, cre]

Maintainer Nick Borcherding <ncborch@gmail.com>

Contents

`immApex-package` *immApex: Tools for Adaptive Immune Receptor Sequence-Based Machine and Deep Learning*

Description

A set of tools to for machine and deep learning in R from amino acid and nucleotide sequences focusing on adaptive immune receptors. The package includes pre-processing of sequences, unifying gene nomenclature usage, encoding sequences, and combining models. This package will serve as the basis of future immune receptor sequence functions/packages/models compatible with the scRepertoire ecosystem.

Author(s)

Maintainer: Nick Borcherding <ncborch@gmail.com>

See Also

Useful links:

- https://github.com/BorchLab/immApex/
- Report bugs at https://github.com/BorchLab/immApex/issues

ace_richness *ACE Richness Estimator*

Description

Calculates the Abundance-based Coverage Estimator (ACE) of species richness. This metric is particularly useful for datasets with a large number of rare species.

Usage

```
ace_richness(cnt)
```

Arguments

cnt	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$S_{ace} = S_{abund} + \frac{S_{rare}}{C_{ace}} + \frac{F_1}{C_{ace}} \gamma_{ace}^2$$

where the classification of rare and abundant species is based on a threshold of 10 individuals, *F*1 is the count of singletons, *S*rare is the number of rare species, and *C*ace is the sample coverage for rare species.

Value

A single numeric value representing the estimated total number of species. The estimate is constrained to be at least the number of observed species.

References

Chao, A., & Lee, S.-M. (1992). *Estimating the number of classes via sample coverage*. Journal of the American Statistical Association, 87(417), 210-217.

Examples

```
counts <- rpois(50, lambda=1.5)
ace_richness(counts)
```

adjacencyMatrix *Adjacency Matrix From Amino Acid or Nucleotide Sequences*

Description

Calculate frequency of adjacency between residues along a set of biological sequences.

Usage

```
adjacencyMatrix(
  input.sequences,
  normalize = TRUE,
  sequence.dictionary = amino.acids,
  directed = FALSE
)
```

Arguments

input.sequences	Character vector of sequences (amino acid or nucleotide)
normalize	Return the values as a normalized frequency (TRUE) or raw counts (FALSE).
sequence.dictionary	The letters to use in the matrix (defaults to a standard 20 amino acids).
directed	Logical; if FALSE (default) the matrix is symmetrised.

Value

An adjacency matrix.

Examples

```
# new.sequences <- generateSequences(prefix.motif = "CAS",
#                                    suffix.motif = "YF",
#                                    number.of.sequences = 100,
#                                    min.length = 8,
#                                    max.length = 16)
#
# adj.matrix <- adjacencyMatrix(new.sequences,
#                               normalize = TRUE)
```

amino.acids *Standard 20 amino acids*

Description

Vector of one-letter codes for the 20 standard amino acids.

Usage

```
amino.acids
```

Format

An object of class `character` of length 20.

`buildNetwork` *Build Edit Distance Network*

Description

Build Edit Distance Network

Usage

```
buildNetwork(
  input.data = NULL,
  input.sequences = NULL,
  seq_col = NULL,
  v_col = NULL,
  j_col = NULL,
  threshold = 2,
  filter.v = FALSE,
  filter.j = FALSE,
  ids = NULL,
  output = c("edges", "sparse"),
  weight = c("dist", "binary")
)
```

Arguments

`input.data` ‘data.frame‘/‘tibble‘ with sequence & metadata (optional - omit if you supply ‘sequences‘ directly).

`input.sequences`
Character vector of sequences **or** column name inside ‘input.data‘. Ignored when ‘NULL‘ and ‘seq_col‘ is non-‘NULL‘.

`seq_col`, `v_col`, `j_col`
Column names to use when ‘input.data‘ is given. By default the function looks for common AIRR names (‘junction_aa‘, ‘cdr3‘, ‘v_call‘, ‘j_call‘).

`threshold` >= 1 for absolute distance **or** 0 < x <= 1 for relative.

`filter.v`, `filter.j`
Logical; require identical V/J when ‘TRUE‘.

`ids` Optional character labels; recycled from row-names if missing.

`output` ‘"edges"‘ (default) or ‘"sparse"‘ - return an edge-list ‘data.frame‘ **or** a symmetric ‘Matrix::dgCMatrix‘ adjacency matrix.

`weight` ‘"dist"‘ (store the edit distance) **or** ‘"binary"‘ (all edges get weight 1). Ignored when ‘output = "edges"‘.

Value

edge-list ‘data.frame‘ **or** sparse adjacency ‘dgCMatrix‘

Examples

```
data(immapex_example.data)

# Build Edge List
edges <- buildNetwork(input.data = immapex_example.data[["AIRR"]],
                      seq_col    = "junction_aa",
                      threshold  = 0.9,
                      filter.v   = TRUE)
```

---

calculateEntropy *Positional Entropy / Diversity Biological Sequences*

Description

Computes residue-wise diversity for a set of aligned (right-padded) CDR3 amino-acid sequences using *any* supported diversity estimator in **immApex**. The following metrics are recognized:

* **Shannon entropy:** shannon_entropy * **Inverse Simpson:** inv_simpson * **Gini–Simpson index:** gini_simpson * **Normalized entropy:** norm_entropy * **Pielou evenness:** pielou_evenness * * **Hill numbers** (orders 0, 1, 2): hill_q(0), hill_q(1), hill_q(2)

You may also supply a **custom function** to ‘method‘; it must take a numeric vector of clone counts and return a single numeric value.

Usage

```
calculateEntropy(
  input.sequences,
  max.length = NULL,
 method = c("shannon", "inv.simpson", "gini.simpson", "norm.entropy", "pielou", "hill0",
    "hill1", "hill2"),
  padding.symbol = "."
)
```

Arguments

input.sequences	‘character()‘. Vector of CDR3 AA strings.
max.length	‘integer(1)‘. Target length to align / pad to. *Default* = ‘max(nchar(sequences))‘.
method	Either the name of a built-in metric (‘"shannon"‘, ‘"inv.simpson"‘, ‘"gini.simpson"‘, ‘"norm.entropy"‘, ‘"pielou"‘, ‘"hill0"‘, ‘"hill1"‘, ‘"hill2"‘) **or** a custom function as described above.
padding.symbol	Symbol to use for padding at the end of sequences.

Value

Named ‘numeric()‘ vector of diversity scores, one value per position (Pos1 … Pos*L*).

Examples

```
seqs <- c("CASSLGQDTQYF", "CASSIRSSYNEQFF", "CASSTGELFF")
calculateEntropy (seqs, method = "shannon")
```

calculateFrequency *Relative Residue Frequencies at Every Position*

Description

Quickly computes the per-position relative frequency of each symbol (amino-acid or nucleotide) in a set of biological sequences. Variable-length strings are padded to a common width so the calculation is entirely vectorized (one logical comparison + one ‘colSums()‘ per residue).

Usage

```
calculateFrequency(
  input.sequences,
  max.length = NULL,
  sequence.dictionary = amino.acids,
  padding.symbol = ".",
  summary.fun = c("proportion", "count", "percent"),
  tidy = FALSE
)
```

Arguments

`input.sequences`
Character vector of sequences (amino acid or nucleotide)

`max.length` Integer. Pad/trim to this length. Defaults to ‘max(nchar(sequences))‘.

`sequence.dictionary`
Vector of valid residue symbols that should be tracked (defaults to the 20 canonical amino acids; supply ‘c("A","C","G","T","N")‘ etc. for nucleotides).

`padding.symbol` Single character used for right-padding. **Must not** be present in ‘sequence.dictionary‘.

`summary.fun` Character string choosing the summary statistic: * ‘"proportion"‘ (default) – each cell sums to 1 over the table. * ‘"count"‘ – raw counts. * ‘"percent"‘ – proportion × 100.

`tidy` Logical; if ‘TRUE‘ a long-format ‘data.frame‘ is returned instead of a matrix (useful for plotting with *ggplot2*).

Value

Either

- A numeric matrix of dimension ‘length(sequence.dictionary)‘ × ‘max.length‘, whose columns sum to 1, **or**
- A ‘data.frame‘ with columns *position*, *residue*, *frequency* when ‘tidy = TRUE‘.

Examples

```
# Amino Acid example
seqs <- c("CASSLGQGAETQYF", "CASSPGQGDYEQYF", "CASSQETQYF")
rel.freq <- calculateFrequency(seqs)
head(rel.freq[, 1:5])
```

```
# Nucleotide example
dna <- c("ATGCC", "ATGAC", "ATGGC")
calculateFrequency(dna,
                   sequence.dictionary = c("A","C","G","T"),
                   padding.symbol = "-",
                   tidy = TRUE)
```

calculateGeneUsage *Quantifcation of Gene-Locus Usage*

Description

Computes either the **counts**, **proportions** (default), or **percentages** of one locus *or* a locus pair that are already present as columns in ‘input.data‘. No external dependencies.

Usage

```
calculateGeneUsage(
  input.data,
  loci,
  levels = NULL,
  summary.fun = c("proportion", "count", "percent")
)
```

Arguments

`input.data`	A data.frame whose rows are sequences / clones and whose columns named in ‘loci‘ contain gene identifiers.
`loci`	Character vector of length 1 or 2 giving the column names.
`levels`	Optional list of length 1 or 2 with the full set of factor levels to include. Missing levels are filled with zeros. If ‘NULL‘ (default) only observed levels appear.
`summary.fun`	Character string choosing the summary statistic: * ‘"proportion"‘ (default) – each cell sums to 1 over the table. * ‘"count"‘ – raw counts. * ‘"percent"‘ – proportion × 100.

Value

Named numeric **vector** (single locus) or numeric **matrix** (paired loci). For ‘"proportion"‘ and ‘"percent"‘ results sum to 1 or 100.

Examples

```
df <- data.frame(V = c("TRBV7-2","TRBV7-2","TRBV5-1"),
                 J = c("TRBJ2-3","TRBJ2-5","TRBJ2-3"))
calculateGeneUsage(df, "V",          summary = "count")
calculateGeneUsage(df, c("V","J"),   summary = "percent")
```

calculateMotif *Motif Enumeration and Counting*

Description

Rapidly enumerates and quantifies **contiguous** (and, optionally, single-gap discontinuous) amino-acid motifs across a set of sequences.

Usage

```
calculateMotif(
  input.sequences,
  motif.lengths = 2:5,
  min.depth = 3,
  discontinuous = FALSE,
  discontinuous.symbol = ".",
  nthreads = 1
)
```

Arguments

`input.sequences`	Character vector of sequences (amino acid or nucleotide)
`motif.lengths`	Integer vector of motif sizes (>= 1). **Default:** ‘2:5‘.
`min.depth`	Minimum count a motif must reach to be retained in the output (‘>= 1‘). **Default:** ‘3‘.
`discontinuous`	Logical; include single-gap motifs as well? **Default:** ‘FALSE‘.
`discontinuous.symbol`	Single character representing the gap when ‘discontinuous = TRUE‘. **Default:** ‘"."‘.
`nthreads`	Integer number of OpenMP threads to use. ‘1‘ forces serial execution. **Default:** ‘1‘.

Details

For every input sequence the algorithm slides windows of length *k* (‘motif.lengths‘) and increments a motif counter (‘unordered_map‘). If ‘discontinuous = TRUE‘, each window is additionally copied *k* times, substituting one position at a time with ‘discontinuous.symbol‘ (default ‘"."‘), yielding gapped motif patterns such as ‘"C.S"‘.

Value

A ‘data.frame‘ with two columns:

motif Motif string (contiguous or gapped).

frequency Integer occurrence count across all sequences.

Examples

```
seqs <- c("CASSLGQDTQYF", "CASSAGQDTQYF", "CASSLGEDTQYF")
calculateMotif(seqs, motif.lengths = 3, min.depth = 2)
```

calculateProperty *Position-wise Amino-Acid Property Profiles*

Description

Computes a range of summary statistics for property values of one or more AA property scales at every residue position of a set of protein (or peptide) sequences. The function is entirely vectorized: it first calls [‘calculateFrequency()‘] to obtain a residue-by-position **frequency** matrix *F* (each column sums to 1) and then performs a single matrix product.

Usage

```
calculateProperty(
  input.sequences,
  property.set = "atchleyFactors",
  summary.fun = "mean",
  transform = "none",
  max.length = NULL,
  padding.symbol = ".",
  tidy = FALSE
)
```

Arguments

`input.sequences`	Character vector of amino-acid strings.
`property.set`	Character string (one of the supported names) Defaults to ‘"atchleyFactors"‘, but includes: ‘"crucianiProperties"‘, ‘"FASGAI"‘, ‘"kideraFactors"‘, ‘"MSWHIM"‘, ‘"ProtFP"‘, ‘"stScales"‘, ‘"tScales"‘, ‘"VHSE"‘, ‘"zScales"‘
`summary.fun`	Character string (‘"mean"‘, ‘"median"‘, ‘"sum"‘, ‘"min"‘, ‘"max"‘), **or** a function accepting a numeric vector and returning length-1 numeric. Defaults to ‘"mean"‘.
`transform`	Character string controlling a *post-summary* transformation. One of ‘"none"‘ (default), ‘"sqrt"‘, ‘"log1p"‘, ‘"zscore"‘ (row-wise), or ‘"minmax"‘ (row-wise).
`max.length`	Integer. Pad/trim to this length (‘max(nchar(sequences))‘ by default).
`padding.symbol`	Single character used for right-padding. Must not be one of the 20 canonical residues.
`tidy`	Logical; if ‘TRUE‘, return a long-format ‘data.frame‘

Value

A numeric matrix (*k* × *L*) **or** a tidy data.frame with columns scale, position, value.

Examples

```
set.seed(1)
seqs <- c("CASSLGQGAETQYF", "CASSPGQGDYEQYF", "CASSQETQYF")
aa.Atchley <- calculateProperty(seqs, property.set = "atchleyFactors")
```

chao1_richness *Chao1 Richness Estimator*

Description

Calculates the Chao1 non-parametric estimator of species richness.

Usage

```
chao1_richness(cnt)
```

Arguments

cnt	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

The bias-corrected formula is used:

$$S_{chao1} = S_{obs} + \frac{F_1(F_1 - 1)}{2(F_2 + 1)}$$

where *S*obs is the number of observed species, *F*1 is the count of singletons, and *F*2 is the count of doubletons.

If the conditions for the formula are not met (*F*1 <= 1 or *F*2 = 0), the function returns the observed richness (*S*obs).

Value

A single numeric value representing the estimated total number of species.

References

Chao, A. (1984). *Nonparametric estimation of the number of classes in a population*. Scandinavian Journal of Statistics, 11(4), 265-270.

Examples

```
# Sample with singletons and doubletons
counts <- c(rep(1, 10), rep(2, 5), 5, 8, 12)
chao1_richness(counts)

# Sample without doubletons returns observed richness
chao1_richness(c(rep(1, 5), 3, 4, 5))
```

d50_dom *D50 Dominance Index*

Description

A convenience wrapper for ‘dxx_dom(cnt, 50)‘. Calculates the minimum number of top clones required to constitute 50

Usage

```
d50_dom(cnt)
```

Arguments

cnt	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Value

The smallest number of categories whose cumulative abundance is at least 50

Examples

```
d50_dom(c(100, 50, 20, 10, 5, rep(1, 5)))
```

dxx_dom *Dxx Dominance Index*

Description

Calculates the minimum number of top clones/sequences (ranked by abundance) that constitute a specified percentage of the total dataset. This function allows the user to designate the percentage.

Usage

```
dxx_dom(cnt, pct)
```

Arguments

cnt	Numeric vector of non-negative counts.
pct	A numeric value (0-100) for the target percentage.

Value

The smallest number of categories whose cumulative abundance is at least ‘pct‘ percent of the total abundance.

See Also

[d50_dom()]

Examples

```
counts <- c(100, 50, 20, 10, 5, rep(1, 5))
dxx_dom(counts, 80)
```

---

formatGenes *Ensure clean gene nomenclature using IMGT annotations*

Description

This function will format the genes into a clean nomenclature using the IMGT conventions.

Usage

```
formatGenes(
  input.data,
  region = "v",
  technology = NULL,
  species = "human",
  simplify.format = TRUE
)
```

Arguments

input.data	Data frame of sequencing data or scRepertoire outputs
region	Sequence gene loci to access - "v", "d", "j", or "c" or a combination using c("v", "d", "j")
technology	The sequencing technology employed - **'TenX'**, **"Adaptive'**, or **'AIRR'**
species	One or two word designation of species. Currently supporting: "human", "mouse", "rat", "rabbit", "rhesus monkey", "sheep", "pig", "platypus", "alpaca", "dog", "chicken", and "ferret"
simplify.format	If applicable, remove the allelic designation (**TRUE**) or retain all information (**FALSE**)

Value

A data frame with the new columns of formatted genes added.

Examples

```
data(immapex_example.data)
formatGenes(immapex_example.data[["TenX"]],
            region = "v",
            technology = "TenX")
```

generateSequences *Randomly Generate Amino Acid Sequences*

Description

Use this to make synthetic amino acid sequences for purposes of testing code, training models, or providing noise.

Usage

```
generateSequences(
  prefix.motif = NULL,
  suffix.motif = NULL,
  number.of.sequences = 100,
  min.length = 1,
  max.length = 10,
  verbose = TRUE,
  sequence.dictionary = amino.acids
)
```

Arguments

`prefix.motif`	A defined amino acid/nucleotide sequence to add to the start of the generated sequences.
`suffix.motif`	A defined amino acid/nucleotide sequence to add to the end of the generated sequences.
`number.of.sequences`	The number of sequences to generate.
`min.length`	The minimum length of the final sequence. If this value is too short to fit the motifs, it will be automatically increased.
`max.length`	The maximum length of the final sequence. If it is less than the final ‘min.length‘, it will also be adjusted.
`verbose`	Logical. If TRUE, prints messages when arguments like ‘min.length‘ or ‘max.length‘ are automatically adjusted.
`sequence.dictionary`	A character vector of the letters to use in random sequence generation.

Value

A character vector of generated sequences.

Examples

```
generateSequences(prefix.motif = "CAS",
                  suffix.motif = "YF",
                  number.of.sequences = 100,
                  min.length = 8,
                  max.length = 16)
```

getIMGT *Get IMGT Sequences for Specific Loci*

Description

Use this to access the ImMunoGeneTics (IMGT) sequences for a specific species and gene loci. More information on IMGT can be found at imgt.org.

Usage

```
getIMGT(
  species = "human",
  chain = "TRB",
  sequence.type = "aa",
  frame = "inframe",
  region = "v",
  max.retries = 3,
  verbose = TRUE
)
```

Arguments

species	One or two-word common designation of species.
chain	Sequence chain to access, e.g., **TRB** or **IGH**.
sequence.type	Type of sequence - **aa** (amino acid) or **nt** (nucleotide).
frame	Designation for **all**, **inframe**, or **inframe+gap**.
region	Gene loci to access.
max.retries	Number of attempts to fetch data in case of failure.
verbose	Print messages corresponding to the processing step.

Value

A list of allele sequences.

Examples

```
## Not run:
TRBV_aa <- getIMGT(species = "human",
                   chain = "TRB",
                   frame = "inframe",
                   region = "v",
                   sequence.type = "aa",
                   max.retries = 3)

## End(Not run)
```

getIR *Extract Immune Receptor Sequences*

Description

Use this to extract immune receptor sequences from a Single-Cell Object or the output of combineTCR and combineBCR.

Usage

```
getIR(
  input.data,
  chains,
  sequence.type = c("aa", "nt"),
  group.by = NULL,
  as.list = FALSE
)
```

Arguments

`input.data`	Single-cell object or the output of combineTCR and combineBCR from scRepertoire
`chains`	Immune Receptor chain to use - **TRA**, **TRB**, **IGH**, or **IGL**
`sequence.type`	Extract amino acid (**aa**) or nucleotide (**nt**) sequences
`group.by`	Optional metadata column (e.g., `"sample.id"`) to group and return results as a named list by that variable.
`as.list`	Logical; if `TRUE`, returns a list split by chain. If `group.by` is also provided, returns a nested list Default is `FALSE`.

Value

A data frame, list of data frames, or nested list of immune receptor sequences depending on `as.list` and `group.by`. Each entry includes CDR3 sequence, V(D)J gene segments, and associated barcodes.

gini_coef *Gini Coefficient of Abundance Inequality*

Description

Calculates the Gini coefficient, a measure of inequality, for a vector of clone/sequence counts. It ranges from 0 (perfect equality) to nearly 1 (maximal inequality).

Usage

```
gini_coef(cnt)
```

Arguments

`cnt` Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$G = \frac{\sum_{i=1}^{S} (2i - S - 1) n_i}{S \sum_{i=1}^{S} n_i}$$

where *n*i are the counts of each of the *S* categories, sorted in non-decreasing order.

Value

A numeric value in [0, 1]. Returns ‘0‘ if there is only one category.

See Also

[gini_simpson()]

Examples

```
# High inequality
gini_coef(c(100, 1, 1, 1))
# Perfect equality
gini_coef(c(10, 10, 10, 10))
```

`gini_simpson` *Gini–Simpson Diversity*

Description

Computes the complement of Simpson’s index (also called the Gini–Simpson index or probability of interspecific encounter):

Usage

```
gini_simpson(cnt)
```

Arguments

`cnt` Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$1 - \lambda = 1 - \sum_{i} p_i^2$$

Value

Value in the interval [0, 1]. Higher numbers indicate greater heterogeneity.

Examples

```
gini_simpson(c(10, 5, 5))
```

hill_q *Hill-Number Generator*

Description

Returns a *function* that computes the Hill diversity of order *q* (also called the “effective number of species”):

Usage

```
hill_q(q)
```

Arguments

q	Numeric order of diversity. Common values: *0* (richness), *1* (exp(*H*)), *2* (inverse Simpson).

Details

$$ {}^{q}D \;=\; \left(\sum_{i} p_i^{q}\right)^{1/(1-q)}, \quad q \neq 1 $$

For *q = 1* the formula is undefined; the limit is

$$ {}^{1}D = e^{H'} $$

.

Value

A **closure**: ‘hill_q(q)‘ returns a function that takes a vector of counts and yields the corresponding ^qD. The returned function is vectorised over its input.

References

Hill, M. O. (1973) *Diversity and Evenness: A Unifying Notation and its Consequences.* Ecology **54** (2), 427–432.

Examples

```
hill1 <- hill_q(1)   # q = 1
hill1(c(5, 1, 1, 1))

hill2 <- hill_q(2)   # q = 2, inverse-Simpson
hill2(c(5, 1, 1, 1))
```

`immapex_blosum.pam.matrices` *List of amino acid substitution matrices*

Description

A list of amino acid substitution matrices, using the Point Accepted Matrix (PAM) and BLOck SUbstitution Matrix (BLOSUM) approaches. A discussion and comparison of these matrices are available at PMID: 21356840.

- BLOSUM45
- BLOSUM50
- BLOSUM62
- BLOSUM80
- BLOSUM100
- PAM30
- PAM40
- PAM70
- PAM120
- PAM250

Usage

```
data("immapex_blosum.pam.matrices")
```

Value

List of 10 substitution matrices

`immapex_example.data` *Example contig data for Apex*

Description

Contains a collection of bulk or paired TCR sequences in the respective formats in the form of a list from the following sources:

- TenX: 10k_Human_DTC_Melanoma_5p_nextgem_Multiplex from 10x Website.
- AIRR: Human_colon_16S8157851 from PMID: 37055623.
- Adaptive: Adaptive_2283_D0 from PMID: 36220826.

More information on the data formats are available: AIRR, Adaptive, and TenX.

Usage

```
data("immapex_example.data")
```

Value

List of 3 example data sets for 10x, AIRR and Adaptive contigs.

immapex_gene.list *A list of IMGT gene names by genes, loci, and species*

Description

A list of regularized gene nomenclature to use for converting for data for uniformity. Data is organize by gene region, loci and species. Not all species are represented in the data and pseudogenes have not been removed.

Usage

```
data("immapex_gene.list")
```

Value

List of gene nomenclature by region, loci, and species.

inferCDR *Infer CDR-loop segments from V-gene calls*

Description

Use this isolate sequences from the CDR loop using the V gene annotation. When there are multiple V gene matches for a single gene, the first allelic sequence is used.

Usage

```
inferCDR(
  input.data,
  reference,
  chain = "TRB",
  technology = c("TenX", "AIRR", "Adaptive", "Omniscope"),
  sequence.type = c("aa", "nt"),
  sequences = c("CDR1", "CDR2"),
  verbose = TRUE
)
```

Arguments

input.data	Data frame output of formatGenes
reference	IMGT reference sequences from getIMGT
chain	Sequence chain to access, like **TRB** or **IGH**
technology	The sequencing technology employed - **TenX**, **Adaptive**, or **AIRR**
sequence.type	Type of sequence - **aa** for amino acid or **nt** for nucleotide
sequences	The specific regions of the CDR loop to get from the data, such as **CDR1**.
verbose	Logical. If ‘TRUE‘ (default), prints a progress message.

Value

A data frame with the new columns of CDR sequences added.

Examples

```
 ## Not run:
# Getting the Sequence Reference
data(immapex_example.data)
TRBV_aa <- getIMGT(species = "human",
                   chain = "TRB",
                   frame = "inframe",
                   region = "v",
                   sequence.type = "aa")

# Ensuring sequences are formatted to IMGT
TenX_formatted <- formatGenes(immapex_example.data[["TenX"]],
                              region = "v",
                              technology = "TenX")

# Inferring CDR loop elements
TenX_formatted <- inferCDR(TenX_formatted,
                           chain = "TRB",
                           reference = TRBV_aa,
                           technology = "TenX",
                           sequence.type = "aa",
                           sequences = c("CDR1", "CDR2"))

## End(Not run)
```

inv_simpson *Inverse Simpson Diversity*

Description

Computes the inverse of Simpson's concentration index, sometimes written as *1/D*. This metric emphasizes dominant categories.

Usage

```
inv_simpson(cnt)
```

Arguments

cnt	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$1/D \;=\; \frac{1}{\sum_i p_i^2}$$

Value

Numeric value >= 1. Equals 1 when all observations belong to a single category.

Examples

```
inv_simpson(c(10, 5, 1))
```

mutateSequences *Randomly Mutate Sequences of Amino Acids*

Description

Use this to mutate or mask sequences for purposes of testing code, training models, or noise.

Usage

```
mutateSequences(
  input.sequences,
  number.of.sequences = 1,
  mutation.rate = 0.01,
  position.start = NULL,
  position.end = NULL,
  sequence.dictionary = amino.acids
)
```

Arguments

`input.sequences`
The amino acid or nucleotide sequences to use

`number.of.sequences`
The number of mutated sequences to return

`mutation.rate` The rate of mutations to introduce into sequences

`position.start` The starting position to mutate along the sequence **Default** = NULL will start the random mutations at position 1

`position.end` The ending position to mutate along the sequence **Default** = NULL will end the random mutations at the last position

`sequence.dictionary`
The letters to use in sequence mutation (default are all amino acids)

Value

A vector of mutated sequences

Examples

```
sequences <- generateSequences(prefix.motif = "CAS",
                               suffix.motif = "YF",
                               number.of.sequences = 100,
                               min.length = 8,
                               max.length = 16)

mutated_sequences <- mutateSequences(sequences,
                                     number.of.sequences = 1,
                                     position.start = 3,
                                     position.end = 8)
```

`norm_entropy` *Normalised Shannon Entropy*

Description

Shannon entropy scaled to the interval [0, 1] by its maximum possible value given *S* observed categories:

Usage

```
norm_entropy(cnt)
```

Arguments

`cnt`	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$H^* = \frac{H'}{\ln S}$$

(also known as “Shannon evenness”).

Value

Numeric value in [0, 1]; ‘0‘ when all observations are in a single category.

Examples

```
norm_entropy(c(40, 10, 10, 10))
```

pielou_evenness *Pielou's Evenness*

Description

Convenience wrapper for normalized Shannon entropy (*E* = *H* / ln *S*).

Usage

```
pielou_evenness(cnt)
```

Arguments

cnt	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Value

Numeric evenness measure in [0, 1].

Examples

```
pielou_evenness(c(3, 3, 3))
```

positionalEncoder *Generate Sinusoidal Positional Encodings*

Description

Creates a matrix of sinusoidal positional encodings as described in the "Attention Is All You Need" paper. This provides a way to inject information about the relative or absolute position of tokens in a sequence.

Usage

```
positionalEncoder(
  max.length = NULL,
  d.model = NULL,
  input.sequences = NULL,
  base = 10000,
  position.offset = 1L
)
```

Arguments

`max.length` The maximum sequence length (number of positions) to encode. This is the primary way to specify the output size.

`d.model` The dimensionality of the embedding. Must be an even number.

`input.sequences` Optional. A character vector of sequences. If provided, ‘max.length‘ is automatically determined from the longest sequence, unless ‘max.length‘ is also explicitly set to a larger value.

`base` The base for the geometric progression of frequencies. The default is 10000, as used in the original paper.

`position.offset` An integer offset for position numbering. Defaults to 1 (1-based indexing common in R). Set to 0 for 0-based indexing.

Value

A matrix of shape ‘max.length‘ x ‘d.model‘ containing the positional encodings.

Details

The implementation uses the standard formulas: ‘PE(pos, 2i) = sin(pos / base^(2i / d.model))‘ ‘PE(pos, 2i+1) = cos(pos / base^(2i / d.model))‘ where ‘pos‘ is the position, ‘i‘ is the dimension pair, ‘d.model‘ is the embedding dimension, and ‘base‘ is a user-definable base, typically 10000.

Examples

```
pos_encoding <- positionalEncoder(max.length = 50,
                                  d.model = 64)

my_sequences <- c("SEQVENCE", "ANOTHERSEQ")
pos_enc_auto <- positionalEncoder(input.sequences = my_sequences,
                                  d.model = 32)
```

`probabilityMatrix` *Position Probability Matrix for Amino Acid or Nucleotide Sequences*

Description

Generates a position-probability (PPM) or position-weight (PWM) matrix from a set of biological sequences.

Usage

```
probabilityMatrix(
  input.sequences,
  max.length = NULL,
  convert.PWM = FALSE,
  background.frequencies = NULL,
  sequence.dictionary = amino.acids,
  pseudocount = 1,
  padding.symbol = "."
)
```

Arguments

`input.sequences`
: Character vector of sequences.

`max.length` Integer; sequences will be right-padded to this length. If NULL (default), pads to the length of the longest sequence in the input.

`convert.PWM` Logical; if TRUE, converts the matrix into a PWM.

`background.frequencies`
: Named vector of background frequencies for PWM calculation. If NULL, a uniform distribution is assumed. Names must correspond to characters in ‘sequence.dictionary‘.

`sequence.dictionary`
: Character vector of residues to include in the matrix.

`pseudocount` A small number added to raw counts for PWM calculation to avoid zero probabilities. Defaults to 1.

`padding.symbol` Single character for right-padding. Must not be in ‘sequence.dictionary‘.

Value

A matrix with position-specific probabilities (PPM) or weights (PWM).

Examples

```
new.sequences <- generateSequences(prefix.motif = "CAS",
                                   suffix.motif = "YF",
                                   number.of.sequences = 100,
                                   min.length = 8,
                                   max.length = 16)

PPM.matrix <- probabilityMatrix(new.sequences)
```

---

scaleMatrix *Fast Matrix Scaling or Transformation*

Description

Applies a chosen transformation to every row *or* column of a numeric matrix without altering its dimensions. Designed for lightweight pre-processing pipelines ahead of machine-learning models.

Usage

```
scaleMatrix(
  x,
 method = c("minmax", "z", "robust_z", "unit_var", "l2", "l1", "sqrt", "log1p", "log2",
    "log10", "arcsinh", "none"),
  margin = 2,
  range = c(0, 1),
  offset = 1e-08,
  cofactor = 5,
  na.rm = TRUE
)
```

Arguments

`x` Numeric matrix (coerced with `as.matrix()`).

`method` Character scalar. One of:

- `"minmax"` – rescale linearly to [`range`].
- `"z"` – mean 0 / sd 1 (per margin).
- `"robust_z"` – median 0 / MAD 1 (outlier-resistant).
- `"unit_var"` – divide by sd (keep mean shifts).
- `"l2"`, `"l1"` – divide by Euclidean / L1 norm.
- `"sqrt"` – element-wise square-root.
- `"log1p"` – element-wise `log1p(x + offset)`.
- `"log2"`, `"log10"` – logs with small offset.
- `"arcsinh"` – `asinh(x / cofactor)` (Flow/CyTOF).
- `"none"` – return unchanged.

`margin` 1 = operate row-wise, 2 = column-wise (default 2).

`range` Numeric length-2 vector for `method = "minmax"`.

`offset` Non-negative scalar added before logs / sqrt (*ignored* otherwise). Default `1e-8`.

`cofactor` Numeric > 0 for `method = "arcsinh"` (default 5).

`na.rm` Logical; drop NAs when computing summaries.

Value

Matrix of identical dimension (dimnames preserved).

Examples

```
m <- matrix(rnorm(20), 4, 5,
            dimnames = list(paste0("g", 1:4), paste0("s", 1:5)))
scaleMatrix(m, "minmax")
scaleMatrix(m, "robust_z", margin = 1)
scaleMatrix(m, "l2")
scaleMatrix(abs(m), "arcsinh", cofactor = 150)
```

`sequenceDecoder` *Decode Amino Acid or Nucleotide Sequences*

Description

Transforms one-hot or property-encoded sequences back into their original character representation. This function serves as the inverse to ‘sequenceEncoder‘.

Usage

```
sequenceDecoder(
  encoded.object,
  mode = c("onehot", "property"),
  property.set = NULL,
  property.matrix = NULL,
  call.threshold = 0.5,
  sequence.dictionary = amino.acids,
  padding.symbol = ".",
  remove.padding = TRUE
)
```

Arguments

`encoded.object` A ‘list‘ object produced by ‘sequenceEncoder‘, or a numeric ‘matrix‘ (flattened 2D) or ‘array‘ (3D cube) from it.

`mode` The encoding mode used for decoding: ‘"onehot"‘ or ‘"property"‘. This is typically inferred if ‘encoded.object‘ is a list from ‘sequenceEncoder‘.

`property.set` For ‘mode = "property"‘, a character vector of property names (e.g., ‘"atchleyFactors"‘) that were used for the original encoding. See ‘?sequenceEncoder‘. This is ignored if ‘property.matrix‘ is supplied.

`property.matrix` For ‘mode = "property"‘, the exact numeric matrix (with dimensions ‘20 x P‘) that was used for encoding. This overrides ‘property.set‘.

`call.threshold` A numeric confidence threshold for making a call. - In ‘"onehot"‘ mode, this is the minimum required value in the vector (e.g., ‘0.9‘). - In ‘"property"‘ mode, this is the maximum allowable Euclidean distance. Positions with scores not meeting the threshold are assigned the ‘padding.symbol‘.

`sequence.dictionary` A character vector of the alphabet (e.g., amino acids). Must match the one used during encoding.

`padding.symbol` The single character used to represent padding or low-confidence positions.

`remove.padding` Logical. If ‘TRUE‘, trailing padding symbols are removed from the end of the decoded sequences.

Value

A character vector of the decoded sequences.

Examples

```
# Example sequences
aa.sequences <- c("CAR", "YMD", "ACAC")

# Encode the sequences
encoded.onehot <- sequenceEncoder(aa.sequences,
                                  mode = "onehot")
encoded.prop <- sequenceEncoder(aa.sequences,
                                mode = "property",
                                property.set = "atchleyFactors")

# Decode the sequences
# 1. Decode from the full list object
decoded.1 <- sequenceDecoder(encoded.onehot,
                             mode = "onehot")

# 2. Decode from just the 3D cube array
decoded.2 <- sequenceDecoder(encoded.prop$cube,
                             mode = "property",
                             property.set = "atchleyFactors")
```

`sequenceEncoder` *Universal Amino-acid Sequence Encoder*

Description

‘sequenceEncoder()‘ is a high-level function that converts a character vector of amino-acid sequences into one of three representations: 1. **one-hot**: A binary representation for each amino acid position. 2. **property-based**: A numerical representation based on amino acid properties (e.g., atchleyFactors, kideraFactors, etc). 3. **geometric**: A fixed-length 20-dimensional vector for each sequence, derived from a substitution matrix and geometric rotation.

Usage

```
sequenceEncoder(
  input.sequences,
  mode = c("onehot", "property", "geometric"),
  property.set = NULL,
  property.matrix = NULL,
  method = "BLOSUM62",
  theta = pi/3,
  sequence.dictionary = amino.acids,
  padding.symbol = ".",
  summary.fun = "",
  max.length = NULL,
  nthreads = parallel::detectCores(),
  verbose = TRUE,
  ...
)

onehotEncoder(..., mode = "onehot")

propertyEncoder(..., mode = "property")

geometricEncoder(..., mode = "geometric")
```

Arguments

`input.sequences`	‘character‘ vector. Sequences (uppercase single-letter code).
`mode`	Either ‘"onehot"‘, ‘"property"‘, or ‘"geometric"‘.
`property.set`	Character string (one of the supported names) Defaults to ‘"atchleyFactors"‘, but includes: ‘"crucianiProperties"‘, ‘"FASGAI"‘, ‘"kideraFactors"‘, ‘"MSWHIM"‘, ‘"ProtFP"‘, ‘"stScales"‘, ‘"tScales"‘, ‘"VHSE"‘, ‘"zScales"‘ Ignored if ‘property.matrix‘ is supplied.
`property.matrix`	*Optional numeric matrix (‘20 × P‘)*. Overrides ‘property.set‘ in ‘"property"‘ mode.

`method` *(For geometric mode)* Character key for a built-in substitution matrix (e.g., "BLOSUM62"), or a 20x20 numeric matrix itself.

`theta` *(For geometric mode)* Rotation angle in radians (default ‘pi/3‘).

`sequence.dictionary`
Character vector of the alphabet (default = 20 standard amino acids).

`padding.symbol` Single character for right-padding (non-geometric modes).

`summary.fun` For property mode only: ‘"mean"‘ or ‘""‘ (none).

`max.length` Integer for truncation/padding. If ‘NULL‘ (default), the longest sequence sets the maximum. Not used in geometric mode.

`nthreads` Number of threads for C++ backend. Not used in geometric mode.

`verbose` Logical. If ‘TRUE‘ (default), prints a progress message.

`...` Additional arguments passed to ‘sequenceEncoder()‘ when using wrapper functions (‘onehotEncoder‘, ‘propertyEncoder‘, ‘geometricEncoder‘).

Details

The function acts as a wrapper for either the C++ backend (for one-hot and property modes) or the R-based geometric transformation.

Value

A named ‘list‘ containing the encoded data and metadata.

‘cube‘ 3D Numeric array. ‘NULL‘ in geometric mode.

‘flattened‘ 2D Numeric matrix. ‘NULL‘ in geometric mode.

‘summary‘ 2D Numeric matrix containing sequence-level representations. This is the primary output for geometric mode.

... Other metadata related to the encoding process.

Property Mode

If you supply ‘property.matrix‘ directly, it **must** be a numeric matrix whose **rows correspond to the 20 canonical amino acids in the order of ‘sequence.dictionary‘** and whose columns are the property scales.

Geometric Mode

This mode projects sequences into a 20D space. It calculates the average vector for each sequence using a substitution matrix (e.g., "BLOSUM62") and then applies a planar rotation to the resulting vector.

Examples

```
aa <- c("CARDRST", "YYYGMD", "ACACACAC")

# One-hot encoding
enc_onehot <- sequenceEncoder(aa,
                              mode = "onehot")

# Property-based encoding
enc_prop <- sequenceEncoder(aa,
                              mode = "property",
                              property.set = "atchleyFactors")

# Geometric encoding
enc_geo <- sequenceEncoder(aa,
                           mode = "geometric",
                           method = "BLOSUM62")
```

`shannon_entropy` *Shannon Diversity Index (Entropy)*

Description

Calculates Shannon's information entropy (often denoted *H*) for a set of clone or sequence counts.

Usage

```
shannon_entropy(cnt)
```

Arguments

`cnt`	Numeric vector of non-negative counts (one entry per clone/ residue/OTU). Zero counts are ignored.

Details

$$H' \;=\; -\sum_{i=1}^{S} p_i \ln p_i$$

where *p*<sub>*i*</sub> = *n*<sub>*i*</sub> / *N* are the relative frequencies (proportions) of each of the *S* distinct categories.

Value

A single numeric value (>= 0). When ‘cnt‘ contains exactly one positive entry the function returns ‘0‘.

See Also

[norm_entropy()], [inv_simpson()]

Examples

```
counts <- c(A = 12, B = 4, C = 4)
shannon_entropy(counts)
```

summaryMatrix *Fast Matrix Summaries*

Description

Computes a comprehensive panel of univariate statistics for every **row** *or* **column** of a numeric matrix. It is designed for lightweight feature-engineering pipelines where many summaries are required up-front (e.g. before modeling).

Usage

```
summaryMatrix(x, margin = 2, stats = "all", na.rm = TRUE)
```

Arguments

x	Numeric matrix (will be coerced with as.matrix()).
margin	Integer. 1 = operate row-wise; 2 = column-wise (default 2).
stats	Character vector naming the statistics to return. Any combination of the following (case-insensitive): • "min" • "max" • "mean" • "median" • "sd" • "var" • "mad" • "sum", • "iqr" • "n" • "na" • "mode" • "all"
na.rm	Logical; ignore NAs when calculating statistics default TRUE).

Value

A numeric matrix with one **row per object that was summarised** (rows of the input when margin = 1, otherwise columns) and one **column per requested statistic**. Row-names (if present) are preserved; column names are the statistic labels.

Examples

```
m <- matrix(rnorm(20), 4, 5,
            dimnames = list(paste0("g", 1:4), paste0("s", 1:5)))

## Column-wise summaries (default)
head(summaryMatrix(m))

## Row-wise summaries
head(summaryMatrix(m, margin = 1))
```

tokenizeSequences *Generate Tokenized Sequences from Amino Acid String*

Description

Use this to transform amino acid sequences into tokens in preparing for deep learning models.

Usage

```
tokenizeSequences(
  input.sequences,
  add.startstop = TRUE,
  start.token = "!",
  stop.token = "^",
  max.length = NULL,
  convert.to.matrix = TRUE,
  padding.symbol = NULL,
  verbose = TRUE
)
```

Arguments

`input.sequences`	The amino acid or nucleotide sequences to use
`add.startstop`	Add start and stop tokens to the sequence
`start.token`	The character to use for the start token
`stop.token`	The character to use for the stop token
`max.length`	Additional length to pad, NULL will pad sequences to the max length of input.sequences
`convert.to.matrix`	Return a matrix (TRUE) or a vector (FALSE)
`padding.symbol`	Single character used for right-padding.
`verbose`	Print messages corresponding to the processing step

Value

Integer matrix (rows = sequences, cols = positions) or list of vectors.

Examples

```
new.sequences <- generateSequences(prefix.motif = "CAS",
                                   suffix.motif = "YF",
                                   number.of.sequences = 100,
                                   min.length = 8,
                                   max.length = 16)

sequence.matrix <- tokenizeSequences(new.sequences,
                                    add.startstop = TRUE,
                                    start.token = "!",
                                    stop.token = "^",
                                    convert.to.matrix = TRUE)
```

`variationalSequences` *Generate Similar Sequences using Variational Autoencoder (Defunct)*

Description

This function is defunct and no longer available.

Usage

```
variationalSequences(...)
```

Details

This function previously generated synthetic sequences using a variational autoencoder (VAE). It has been removed for maintenance and clarity.

Value

No return value, called for side effects only.

Index