Package 'immApex'

May 7, 2025

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

Version 1.2.0

Description A set of tools to build tensorflow/keras3-based models in R from amino acid and nucleotide sequences focusing on adaptive immune receptors. The package includes preprocessing 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, igraph, keras3, magrittr, matrixStats, methods, Rcpp (>= 0.12.11), reticulate, rvest, SingleCellExperiment, stats, stringi, stringr, tensorflow, utils

Suggests BiocStyle, ggraph, ggplot2, knitr, graph, markdown, rmarkdown, scRepertoire, spelling, testthat, tidygraph, viridis

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 RELEASE_3_21

git_last_commit 15c8cc3

git_last_commit_date 2025-05-05

Repository Bioconductor 3.21

Date/Publication 2025-05-07

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 build tensorflow/keras3-based models 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

`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 = NULL,
  normalize = TRUE,
  sequence.dictionary = amino.acids
)
```

Arguments

`input.sequences`	The amino acid or nucleotide sequences to use
`normalize`	Return the values as a function of total number of residues (**TRUE**) or frequencies (**FALSE**)
`sequence.dictionary`	The letters to use in sequence generation (default are all amino acids)

Value

Adjacency matrix based on input.sequences.

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)
```

`buildNetwork` *Build Edit Distance Network Using Symmetric Deletion Lookup*

Description

Constructs a weighted similarity network from biological sequences using a symmetric deletion lookup strategy combined with a banded edit-distance computation. The returned igraph object contains vertices representing the input sequences and edges representing pairs of sequences whose edit distance is less than or equal to the specified threshold. The edge attribute `weight` stores the computed edit distance.

Usage

```
buildNetwork(
  input.data,
  sequence.column = "sequence",
  threshold = 2,
  filter.v = FALSE,
  filter.j = FALSE,
  technology = NULL,
  simplify.format = TRUE,
  simplify.families = TRUE
)
```

Arguments

`input.data`	A character vector of AIR sequences, or a data frame containing sequence data.
`sequence.column`	A character string specifying the name of the column in `input.data` that contains the sequences. Default is `"sequence"`. This parameter is ignored when `input.data` is a character vector.
`threshold`	An integer specifying the maximum allowed edit distance. Only pairs of sequences with an edit distance less than or equal to this value will be connected. Default is `2`.
`filter.v`	Logical indicating whether to filter candidate pairs to only those that have matching `v.gene` family annotations. Default is `FALSE`. When `TRUE`, the input data frame must contain a column with V gene annotations, either named `v.gene` or determined by `.get.genes.updated`.
`filter.j`	Logical indicating whether to filter candidate pairs to only those that have matching `j.gene` family annotations. Default is `FALSE`. When `TRUE`, the input data frame must contain a column with J gene annotations, either named `j.gene` or determined by `.get.genes.updated`.
`technology`	The sequencing technology employed - **'TenX'**, **'Adaptive'**, or **'AIRR'**.
`simplify.format`	If applicable, remove the allelic designation (**TRUE**) or retain all information (**FALSE**)

simplify.families
: If applicable, remove the hyphenated designation (**TRUE**) or retain all information (**FALSE**)

Details

This function supports both a character vector of sequences and a data frame. When provided a data frame, the user can specify the column containing sequences using the `sequence.column` parameter. Additionally, candidate pairs can be filtered by requiring matching `v.gene` and/or `j.gene` annotations (see `filter.v` and `filter.j`). If filtering is enabled, the corresponding gene annotation columns are required.

The function first calls a C++ routine (via Rcpp) to perform a symmetric deletion lookup, generating candidate pairs of sequences that might be within the specified edit distance. It then uses a banded dynamic programming algorithm (also implemented in C++) to compute the exact edit distance for each candidate pair. When using a data frame input, the candidate pairs can be further filtered by requiring that sequences have matching `v.gene` and/or `j.gene` values. Note that gene filtering is only applied if the corresponding filtering flag is set to `TRUE`.

Value

An igraph object representing the AIR similarity network. Vertices contain the original sequences (and gene annotations, if available), and each edge has a `weight` attribute corresponding to the computed edit distance. If no edges meet the threshold, an igraph object with only vertices is returned.

Examples

```
# Using a character vector of sequences:
sequences <- c("CASSLGTDTQYF", "CASSPGTDTQYF", "CASSLGNDTQYF", "CASRLGNDTQYF")
g <- buildNetwork(sequences, threshold = 2)
plot(g)

# Using a data frame with a custom sequence column:
df <- data.frame(
  mySeqs = c("CASSLGTDTQYF", "CASSPGTDTQYF", "CASSLGNDTQYF", "CASRLGNDTQYF"),
  v = c("TRBV20", "TRBV20", "TRBV12", "TRBV20"),
  j = c("TRBJ2-7", "TRBJ2-7", "TRBJ2-1", "TRBJ2-7")
)
g_df <- buildNetwork(df,
                     threshold = 2,
                     filter.v = TRUE,
                     filter.j = TRUE,
                     sequence.column = "mySeqs")
plot(g_df)
```

`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 noise.

Usage

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

Arguments

prefix.motif	Add defined amino acid/nucleotide sequence to the start of the generated sequences.
suffix.motif	Add defined amino acid/nucleotide sequence to the end of the generated sequences
number.of.sequences	Number of sequences to generate
min.length	Minimum length of the final sequence. The min.length may be adjusted if incongruent with prefix.motif/suffix.motif lengths
max.length	Maximum length of the final sequence
sequence.dictionary	The letters to use in sequence generation (default are all amino acids)

Value

A vector of generated sequences

Examples

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

`geometricEncoder` *Geometric Encoder from Amino Acid Strings*

Description

Use this to transform amino acid sequences into a geometric encoding of the sequence.

Usage

```
geometricEncoder(
  input.sequences,
  method.to.use = "BLOSUM62",
  theta = pi/3,
  verbose = TRUE
)
```

Arguments

`input.sequences`	The set of amino acid sequences
`method.to.use`	The method or approach for the conversion: • BLOcks SUbstitution Matrices: BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, BLOSUM100 • Point Accepted Mutation Matrices: PAM30, PAM40, PAM70, PAM120, PAM250
`theta`	The angle for geometric transformation
`verbose`	Print messages corresponding to the processing step

Value

Geometric encoded amino acid sequences in a matrix

Examples

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

sequence.matrix <- geometricEncoder(new.sequences,
                                    method.to.use = "BLOSUM62",
                                    theta = pi/3)
```

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

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

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 = "aa")
```

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

Value

A data frame of nucleotide or amino acid sequences

immapex_AA.data *A list of amino acid properties*

Description

A list of amino acid properties that are used for `propertyEncoder` function.

This includes:

- atchleyFactors
- crucianiProperties
- FASGAI
- kideraFactors
- MSWHIM
- ProtFP
- stScales
- tScales
- VHSE
- zScales

Usage

```
data("immapex_AA.data")
```

Value

List of 10 amino acid properties for 20 amino acids

---

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 portions of the CDR Loop from Vgene data*

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 = NULL,
  chain = "TRB",
  technology = NULL,
  sequence.type = "aa",
  sequences = c("CDR1", "CDR2")
)
```

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**, **AIRR**, or **Omniscope**
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**.

Value

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

Examples

```
# 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"))
```

`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,
  n.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
`n.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,
                                     n.sequence = 1,
                                     position.start = 3,
                                     position.end = 8)
```

---

onehotEncoder *One Hot Encoder from Amino Acid or Nucleotide Strings*

Description

Use this to transform amino acid or nucleotide sequences into a one hot encoding of the sequence.

Usage

```
onehotEncoder(
  input.sequences,
  max.length = NULL,
  motif.length = 1,
  convert.to.matrix = TRUE,
  sequence.dictionary = amino.acids,
  padding.symbol = ".",
  verbose = TRUE
)
```

Arguments

input.sequences	The amino acid or nucleotide sequences to use
max.length	Additional length to pad, NULL will pad sequences to the max length of input.sequences
motif.length	The length of the amino acid residues to encode - a motif.length = 1 produces single amino acid encodings
convert.to.matrix	Return a matrix (**TRUE**) or a 3D array (**FALSE**)
sequence.dictionary	The letters to use in sequence generation (default are all amino acids). This will be overrode if using a motif approach (**motif.length** > 1)
padding.symbol	Symbol to use for padding at the end of sequences
verbose	Print messages corresponding to the processing step

Value

One hot encoded sequences in a matrix or 3D array

Examples

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

if(reticulate::py_module_available("numpy")) {
sequence.matrix <- onehotEncoder(new.sequences,
                                 convert.to.matrix = TRUE)
}
```

`positionalEncoder` *Adding Position-Specific Information to Sequences*

Description

Use this calculate positional encoding for recurrent neural networks using sin/cos and position information.

Usage

```
positionalEncoder(number.of.sequences, latent.dims = NULL)
```

Arguments

`number.of.sequences`	The number of sequences to generate position information
`latent.dims`	The number of latent dimensions.

Value

A matrix of values

Examples

```
position.info <- positionalEncoder(number.of.sequences = 1000,
                                   latent.dims = 64)
```

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

Description

Use this to generate a position-probability or weight matrix for a set of given sequences.

Usage

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

Arguments

input.sequences	The amino acid or nucleotide sequences to use
max.length	Additional length to pad, NULL will pad sequences to the max length of input.sequences
convert.PWM	Convert the matrix into a positional weight matrix using log likelihood
background.frequencies	Provide amino acid or nucleotide frequencies for the positional weight matrix. If NULL, assumes uniform likelihood.
sequence.dictionary	The letters to use in sequence generation (default are all amino acids)
padding.symbol	Symbol to use for padding at the end of sequences
verbose	Print messages corresponding to the processing step

Value

A matrix with position specific probabilities or weights

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)
```

propertyEncoder *Encoder from Amino Acid String by Properties*

Description

Use this to transform amino acid sequences a a matrix by amino acid properties derived from dimensional reduction strategies

Usage

```
propertyEncoder(
  input.sequences,
  max.length = NULL,
  method.to.use = NULL,
  convert.to.matrix = TRUE,
  summary.function = NULL,
  padding.symbol = ".",
  verbose = TRUE
)
```

Arguments

input.sequences	The amino acid sequences to use
max.length	Additional length to pad, NULL will pad sequences to the max length of input.sequences
method.to.use	The method or approach to use for the conversion: • Individual sets: atchleyFactors, crucianiProperties, FASGAI, kideraFactors, MSWHIM, ProtFP, stScales, tScales, VHSE, zScales" • Multiple Sets: c("atchleyFactors", "VHSE")
convert.to.matrix	Return a matrix (TRUE) or a 3D array (FALSE)
summary.function	Return a matrix that summarize the amino acid method/property Available summaries include: "median", "mean", "sum", variance ("vars"), or Median Absolute Deviation ("mads")
padding.symbol	Symbol to use for padding at the end of sequences
verbose	Print messages corresponding to the processing step

Value

Converted amino acid sequences by property in a matrix or 3D array

Examples

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

if(reticulate::py_module_available("numpy")) {
sequence.matrix <- propertyEncoder(new.sequences,
                                   method.to.use = "VHSE",
                                   convert.to.matrix = TRUE)
}
```

---

sequenceDecoder *One Hot Decoder from One Hot Encoded Matrix or 3D Array*

Description

Use this to transform one hot encoded sequences back into amino acid or nucleotide sequences.

Usage

```
sequenceDecoder(
  sequence.matrix,
  encoder = "onehotEncoder",
  aa.method.to.use = NULL,
  call.threshold = 0.5,
  sequence.dictionary = amino.acids,
  padding.symbol = ".",
  remove.padding = TRUE
)
```

Arguments

`sequence.matrix`
: The encoded sequences to decode in an array or matrix

`encoder`
: The method to prepare the sequencing information - "onehotEncoder" or "propertyEncoder"

`aa.method.to.use`
: The method or approach to use for the conversion:

- Individual sets: atchleyFactors, crucianiProperties, FASGAI, kideraFactors, MSWHIM, ProtFP, stScales, tScales, VHSE, zScales"
- Multiple Sets: c("atchleyFactors", "VHSE")

`call.threshold`
: The relative strictness of sequence calling with higher values being more stringent

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

`padding.symbol`
: Symbol to use for padding at the end of sequences

`remove.padding`
: Remove the additional symbol from the end of decoded sequences

Value

Decoded amino acid or nucleotide sequences

Examples

```
new.sequences <- generateSequences(prefix.motif = "CAS",
                                   suffix.motif = "YF",
                                   number.of.sequences = 100,
                                   min.length = 8,
                                   max.length = 16)
if(reticulate::py_module_available("numpy")) {
sequence.matrix <- onehotEncoder(new.sequences,
                                 convert.to.matrix = TRUE)

decoded.sequences <- sequenceDecoder(sequence.matrix,
                                     padding.symbol = ".")
}
```

---

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,
  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)
verbose	Print messages corresponding to the processing step

Value

Tokenize sequences in a matrix or vector

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*

Description

Use this to simulate sequences using a variational autoencoder (VAE) and perturbation of the probability distributions.

Usage

```
variationalSequences(
  input.sequences,
  encoder.function = "onehotEncoder",
  aa.method.to.use = NULL,
  number.of.sequences = 100,
```

```
  encoder.hidden.dim = c(128, 64),
  decoder.hidden.dim = NULL,
  latent.dim = 16,
  batch.size = 16,
  epochs = 50,
  learning.rate = 0.001,
  epsilon.std = 1,
  call.threshold = 0.2,
  activation.function = "relu",
  optimizer = "adam",
  disable.eager.execution = FALSE,
  sequence.dictionary = amino.acids,
  verbose = TRUE
)
```

Arguments

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

`encoder.function`
: The method to prepare the sequencing information - "onehotEncoder" or "propertyEncoder"

`aa.method.to.use`
: The method or approach to use for the conversion:
  - Individual sets: atchleyFactors, crucianiProperties, FASGAI, kideraFactors, MSWHIM, ProtFP, stScales, tScales, VHSE, zScales"
  - Multiple Sets: c("atchleyFactors", "VHSE")

`number.of.sequences`
: Number of sequences to generate

`encoder.hidden.dim`
: A vector of the neurons to use in the hidden layers for the encoder portion of the model

`decoder.hidden.dim`
: A vector of the neurons to use in the hidden layers for the decoder portion of the model. If NULL assumes symmetric autoencoder

`latent.dim` The size of the latent dimensions

`batch.size` The batch size to use for VAE training

`epochs` The number of epochs to use in VAE training

`learning.rate` The learning rate to use in VAE training

`epsilon.std` The epsilon to use in VAE training

`call.threshold` The relative strictness of sequence calling with higher values being more stringent

`activation.function`
: The activation for the dense connected layers

`optimizer` The optimizer to use in VAE training

`disable.eager.execution`
: Disable the eager execution parameter for tensorflow.

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

`verbose`
: Print messages corresponding to the processing step

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)

if(reticulate::py_module_available("keras")) {
new.sequences <- variationalSequences(sequences,
                                      encoder = "onehotEncoder",
                                      encoder.hidden.dim = c(256, 128),
                                      latent.dim = 16,
                                      batch.size = 16)
}
```

Index