

Package ‘HiLDA’

November 30, 2020

Type Package

Title Conducting statistical inference on comparing the mutational exposures of mutational signatures by using hierarchical latent Dirichlet allocation

Depends R(>= 3.6), ggplot2

Imports R2jags, abind, cowplot, grid, forcats, stringr, GenomicRanges, S4Vectors, XVector, Biostrings, GenomicFeatures, BSgenome.Hsapiens.UCSC.hg19, BiocGenerics, tidyr, grDevices, stats, TxDb.Hsapiens.UCSC.hg19.knownGene, utils, methods, Rcpp

Suggests knitr, rmarkdown, testthat, BiocStyle

Version 1.4.0

Date 2019-09-10

Description A package built under the Bayesian framework of applying hierarchical latent Dirichlet allocation to statistically test whether the mutational exposures of mutational signatures (Shiraishi-model signatures) are different between two groups.

License GPL-3

URL <https://github.com/USCbiostats/HiLDA>,
<https://doi.org/10.1101/577452>

BugReports <https://github.com/USCbiostats/HiLDA/issues>

SystemRequirements JAGS 4.2.0

biocViews Software, SomaticMutation, Sequencing, StatisticalMethod, Bayesian

RoxygenNote 6.1.1

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

git_url <https://git.bioconductor.org/packages/HiLDA>

git_branch RELEASE_3_12

git_last_commit f8470d0

git_last_commit_date 2020-10-27

Date/Publication 2020-11-29

Author Zhi Yang [aut, cre],
Yuichi Shiraishi [ctb]

Maintainer Zhi Yang <zyang895@gmail.com>

R topics documented:

boundaryTurbo_F	2
boundaryTurbo_Q	3
calcPMSLikelihood	3
convertFromTurbo_F	4
convertFromTurbo_Q	4
convertToTurbo_F	5
convertToTurbo_Q	5
EstimatedParameters-class	6
getLogLikelihoodC	6
getMutationFeatureVector	7
hildaBarplot	7
hildaDiffPlot	8
hildaGlobalResult	9
hildaLocalResult	9
hildaPlotSignature	10
hildaReadMPFile	11
hildaRhat	12
hildaTest	12
MetaInformation-class	13
MutationFeatureData-class	14
mySquareEM	14
pmBarplot	15
pmgetSignature	16
pmMultiBarplot	16
pmPlotSignature	17
PMSboundary	18
updateMstepFQC	18
updatePMSParam	19
updateTheta_NormalizedC	19
visPMS	20
Index	22

boundaryTurbo_F

Check whether the parameter F is within the appropriate range

Description

Check whether the parameter F is within the appropriate range

Usage

boundaryTurbo_F(turboF, fdim, signatureNum)

Arguments

turboF	F (converted for turboEM)
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures

Value

a logical value

boundaryTurbo_Q	<i>Check whether the parameter Q is within the appropriate range</i>
-----------------	----------------------------------------------------------------------

Description

Check whether the parameter Q is within the appropriate range

Usage

```
boundaryTurbo_Q(turboQ, signatureNum, sampleNum)
```

Arguments

turboQ	Q (converted for turboEM)
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes

Value

a logical value

calcPMSLikelihood	<i>A function for calculating the log-likelihood from the data and parameters</i>
-------------------	-----------------------------------------------------------------------------------

Description

A function for calculating the log-likelihood from the data and parameters

Usage

```
calcPMSLikelihood(p, y)
```

Arguments

p	this variable includes the parameters for mutation signatures and membership parameters
y	this variable includes the information on the mutation features, the number of mutation signatures specified and so on

Value

a value

convertFromTurbo_F *Restore the converted parameter F for turboEM*

Description

Restore the converted parameter F for turboEM

Usage

```
convertFromTurbo_F(turboF, fdim, signatureNum, isBackground)
```

Arguments

turboF	F (converted for turboEM)
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures
isBackground	the logical value showing whether a background mutaiton features is included or not

Value

a vector

convertFromTurbo_Q *Restore the converted parameter Q for turboEM*

Description

Restore the converted parameter Q for turboEM

Usage

```
convertFromTurbo_Q(turboQ, signatureNum, sampleNum)
```

Arguments

turboQ	Q (converted for turboEM)
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes

Value

a vector

convertToTurbo_F *Convert the parameter F so that turboEM can treat*

Description

Convert the parameter F so that turboEM can treat

Usage

convertToTurbo_F(vF, fdim, signatureNum, isBackground)

Arguments

vF	F (converted to a vector)
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures
isBackground	the logical value showing whether a background mutaiton features is included or not

Value

a vector

convertToTurbo_Q *Convert the parameter Q so that turboEM can treat*

Description

Convert the parameter Q so that turboEM can treat

Usage

convertToTurbo_Q(vQ, signatureNum, sampleNum)

Arguments

vQ	Q (converted to a vector)
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes

Value

a vector

EstimatedParameters-class

An S4 class representing the estimated parameters

Description

An S4 class representing the estimated parameters

Slots

sampleList a list of sample names observed in the input mutation data
signatureNum the number of mutation signatures specified at the time of estimation
isBackGround the flag showing whether the background signature data is used or not.
backGroundProb the background signatures
signatureFeatureDistribution estimated parameters for mutation signatures
sampleSignatureDistribution estimated parameters for memberships of mutation signatures for each sample
loglikelihood the log-likelihood value for the estimated parameters

getLogLikelihoodC

Calculate the value of the log-likelihood for given parameters

Description

Calculate the value of the log-likelihood for given parameters

Usage

```
getLogLikelihoodC(vPatternList, vSparseCount, vF, vQ, fdim, signatureNum,
  sampleNum, patternNum, samplePatternNum, isBackground, vF0)
```

Arguments

vPatternList	The list of possible mutation features (converted to a vector)
vSparseCount	The table showing (mutation feature, sample, the number of mutation) (converted to a vector)
vF	F (converted to a vector)
vQ	Q (converted to a vector)
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes
patternNum	the number of possible combinations of all the mutation features
samplePatternNum	the number of possible combination of samples and mutation patterns
isBackground	the logical value showing whether a background mutation features is included or not
vF0	a background mutation features

Value

a value

getMutationFeatureVector

Get mutation feature vector from context sequence data and reference and alternate allele information

Description

Get mutation feature vector from context sequence data and reference and alternate allele information

Usage

```
getMutationFeatureVector(context, ref_base, alt_base, strandInfo = NULL,
  numBases, type)
```

Arguments

context	the context sequence data around the mutated position. This should be Biostrings::DNASTringSet class
ref_base	the reference bases at the mutated position.
alt_base	the alternate bases at the mutated position.
strandInfo	transcribed strand information at the mutated position. (this is optional)
numBases	the number of flanking bases around the mutated position.
type	the type of mutation feature vecotr (should be "independent" or "full").

Value

a mutation featurer vector

hildaBarplot

Read the raw mutation data with the mutation feature vector format, estimate and plot both mutation signatures and their fractions

Description

Read the raw mutation data with the mutation feature vector format, estimate and plot both mutation signatures and their fractions

Usage

```
hildaBarplot(inputG, hildaResult, sigOrder = NULL, refGroup,
  sortSampleNum = TRUE, refName = "Control", altName = "Case",
  charSize = 3)
```

Arguments

<code>inputG</code>	a <code>MutationFeatureData</code> S4 class output by the <code>pmsignature</code> .
<code>hildaResult</code>	a <code>rjags</code> class output by <code>HiLDA</code> .
<code>sigOrder</code>	the order of signatures if needed (default: <code>NULL</code>).
<code>refGroup</code>	the samples in the reference group (default: <code>NULL</code>).
<code>sortSampleNum</code>	whether to sort plots by number of mutations (default: <code>TRUE</code>).
<code>refName</code>	the name of reference group (default: <code>Control</code>)
<code>altName</code>	the name of the other group (default: <code>Case</code>)
<code>charSize</code>	the size of the character on the signature plot (default: 3)

Value

a list of a signature plot and a barplot of mutational exposures

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
inputFile <- system.file("extdata/hildaLocal.rdata", package="HiLDA")
hildaLocal <- readRDS(inputFile)

hildaBarplot(G, hildaLocal, refGroup=1:4)
```

<code>hildaDiffPlot</code>	<i>Read the raw mutation data with the mutation feature vector format, estimate and plot both mutation signatures and their fractions</i>
----------------------------	-------------------------------------------------------------------------------------------------------------------------------------------

Description

Read the raw mutation data with the mutation feature vector format, estimate and plot both mutation signatures and their fractions

Usage

```
hildaDiffPlot(inputG, hildaResult, sigOrder = NULL, charSize = 3)
```

Arguments

<code>inputG</code>	a <code>MutationFeatureData</code> S4 class output by the <code>pmsignature</code> .
<code>hildaResult</code>	a <code>rjags</code> class output by <code>HiLDA</code> .
<code>sigOrder</code>	the order of signatures if needed (default: <code>NULL</code>).
<code>charSize</code>	the size of the character on the signature plot (default: 3)

Value

a list of the signature plot and the mean difference plot.

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
inputFile <- system.file("extdata/hildaLocal.rdata", package="HiLDA")
hildaLocal <- readRDS(inputFile)

hildaDiffPlot(G, hildaLocal)
```

hildaGlobalResult	<i>Compute the Bayes factor</i>
-------------------	---------------------------------

Description

Compute the Bayes factor

Usage

```
hildaGlobalResult(jagsOutput, pM1 = 0.5)
```

Arguments

jagsOutput	the output jags file generated by the jags function from the R2jags package.
pM1	the probability of sampling the null (default: 0.5)

Value

a number for the Bayes factor

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
hildaGlobal <- hildaTest(inputG=G, numSig=3, refGroup=1:4, nIter=1000,
  localTest=TRUE)
hildaGlobalResult(hildaGlobal)
```

hildaLocalResult	<i>Extract the posterior distributions of the mean differences in muational exposures</i>
------------------	-------------------------------------------------------------------------------------------

Description

Extract the posterior distributions of the mean differences in muational exposures

Usage

```
hildaLocalResult(jagsOutput)
```

Arguments

`jagsOutput` the output jags file generated by the `jags` function from the `R2jags` package.

Value

a data frame that contains the posterior distributions of difference.

Examples

```
inputFile <- system.file("extdata/hildaLocal.rdata", package="HiLDA")
hildaLocal <- readRDS(inputFile)
hildaLocalResult(hildaLocal)
```

`hildaPlotSignature` *Plot mutation signatures from HiLDA output*

Description

Plot mutation signatures from HiLDA output

Usage

```
hildaPlotSignature(hildaResult, sigOrder = NULL, colorList = NULL, ...)
```

Arguments

`hildaResult` a `rjags` class output by HiLDA
`sigOrder` the order of signatures if needed (default: `NULL`)
`colorList` a vector of color for mutational exposures barplots
`...` additional arguments passed on to `visPMS`

Value

a plot object containing all mutational signatures

Examples

```
inputFile <- system.file("extdata/hildaLocal.rdata", package="HiLDA")
hildaLocal <- readRDS(inputFile)
hildaPlotSignature(hildaLocal)
```

hildaReadMPFile	<i>Read the raw mutation data of Mutation Position Format.</i>
-----------------	----------------------------------------------------------------

Description

The mutation position format is tab-delimited text file, where the 1st-5th columns shows sample names, chromosome names, coordinates, reference bases (A, C, G, or T) and the alternate bases (A, C, G, or T), respectively. An example is as follows;

```
—
sample1 chr1 100 A C
sample1 chr1 200 A T
sample1 chr2 100 G T
sample2 chr1 300 T C
sample3 chr3 400 T C
—
```

Also, this function usually can accept compressed files (e.g., by gzip, bzip2 and so on) when using recent version of R.

Usage

```
hildaReadMPFile(infile, numBases = 3, trDir = FALSE,
  bs_genome = NULL, txdb_transcript = NULL)
```

Arguments

infile	the path for the input file for the mutation data of Mutation Position Format.
numBases	the number of upstream and downstream flanking bases (including the mutated base) to take into account.
trDir	the index representing whether transcription direction is considered or not. The gene annotation information is given by UCSC knownGene (TxDb.Hsapiens.UCSC.hg19.knownGene object) When trDir is TRUE, the mutations located in intergenic region are excluded from the analysis.
bs_genome	this argument specifies the reference genome (e.g., B Sgenome.Mmusculus.UCSC.mm10 can be used for the mouse genome). See https://bioconductor.org/packages/release/bioc/html/BSgenome for the available genome list
txdb_transcript	this argument specified the transcript database (e.g., TxDb.Mmusculus.UCSC.mm10.knownGene can be used for the mouse genome). See https://bioconductor.org/packages/release/bioc/html/AnnotationDb for details.

Value

The output is an instance of MutationFeatureData S4 class (which stores summarized information on mutation data). This will be typically used as the initial values for the global test and the local test.

Examples

```
inputFile <- system.file("extdata/esophageal.mp.txt.gz", package="HiLDA")
G <- hildaReadMPFile(inputFile, numBases=5, trDir=TRUE)
```

hildaRhat	<i>Output the maximum potential scale reduction statistic of all parameters estimated</i>
-----------	-------------------------------------------------------------------------------------------

Description

Output the maximum potential scale reduction statistic of all parameters estimated

Usage

```
hildaRhat(jagsOutput)
```

Arguments

jagsOutput the output jags file generated by the jags function from the R2jags package.

Value

a number for the Rhat statistic.

Examples

```
inputFile <- system.file("extdata/hildaLocal.rdata", package="HiLDA")
hildaLocal <- readRDS(inputFile)
hildaRhat(hildaLocal)
```

hildaTest	<i>Apply HiLDA to statistically testing the global difference in burdens of mutation signatures between two groups</i>
-----------	------------------------------------------------------------------------------------------------------------------------

Description

Apply HiLDA to statistically testing the global difference in burdens of mutation signatures between two groups

Usage

```
hildaTest(inputG, numSig, refGroup, useInits = NULL, sigOrder = NULL,
  nIter = 2000, nBurnin = 0, pM1 = 0.5, localTest = TRUE, ...)
```

Arguments

inputG	a MutationFeatureData S4 class output by the pmsignature.
numSig	an integer number of the number of mutational signatures.
refGroup	the indice indicating the samples in the reference group.
useInits	a EstimatedParameters S4 class output by the pmsignature (default: NULL)
sigOrder	the order of the mutational signatures.
nIter	number of total iterations per chain (default: 2000).
nBurnin	length of burn (default: 0).
pM1	the probability of sampling the null (default: 0.5)
localTest	a logical value (default: TRUE)
...	Other arguments passed on to methods.

Value

the output jags file

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))

## with initial values
hildaLocal <- hildaTest(inputG=G, numSig=3, refGroup=1:4, nIter=1000,
  localTest=TRUE)
hildaGlobal <- hildaTest(inputG=G, numSig=3, refGroup=1:4, nIter=1000,
  localTest=FALSE)
```

MetaInformation-class *An S4 class to represent a mutation meta information common to many data types*

Description

@slot type type of data format (independent, full, custom) @slot flankingBasesNum the number of flanking bases to consider (only applicable for independent and full types) @slot transcriptionDirection the flag representing whether transcription direction is considered or not @slot possibleFeatures a vector representing the numbers of possible values for each mutation feature

MutationFeatureData-class

An S4 class representing the mutation data

Description

An S4 class representing the mutation data

Slots

`featureVectorList` a list of feature vectors actually observed in the input mutation data

`sampleList` a list of sample names observed in the input mutation data

`countData` a matrix representing the number of mutations and samples. The (1st, 2nd, 3rd) columns are for (mutation pattern index, sample index, frequencies).

`mutationPosition` a data frame containing position and mutations

mySquareEM

A function for estimating parameters using Squared EM algorithm

Description

A function for estimating parameters using Squared EM algorithm

Usage

```
mySquareEM(p, y, tol = 1e-04, maxIter = 10000)
```

Arguments

`p` this variable includes the parameters for mutation signatures and membership parameters

`y` this variable includes the information on the mutation features, the number of mutation signatures specified and so on

`tol` tolerance for the estimation (when the difference of log-likelihoods become below this value, stop the estimation)

`maxIter` the maximum number of iteration of estimation

Value

a list

pmBarplot	<i>Plot both mutation signatures and their mutational exposures from pm-signature output</i>
-----------	----------------------------------------------------------------------------------------------

Description

Plot both mutation signatures and their mutational exposures from pmsignature output

Usage

```
pmBarplot(inputG, inputParam, sigOrder = NULL, refGroup = NULL,  
          sortSampleNum = TRUE, refName = "Control", altName = "Case",  
          charSize = 3)
```

Arguments

inputG	a MutationFeatureData S4 class output by the pmsignature.
inputParam	a estimatedParameters S4 class output by the pmsignature.
sigOrder	the order of signatures if needed (default: NULL).
refGroup	the samples in the reference group (default: NULL).
sortSampleNum	whether to sort by number of mutations (default: TRUE).
refName	the name of reference group (default: Control).
altName	the name of the other group (default: Case).
charSize	the size of the character on the signature plot (default: 3).

Value

a list of a signature plot and a barplot of mutational exposures

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))  
Param <- pmgetSignature(G, K = 3)  
  
pmPlots <- pmBarplot(G, Param, refGroup=1:4)  
cowplot::plot_grid(pmPlots$sigPlot, pmPlots$propPlot, rel_widths = c(1,3))
```

pmgetSignature	<i>Obtain the parameters for mutation signatures and memberships</i>
----------------	----------------------------------------------------------------------

Description

Obtain the parameters for mutation signatures and memberships

Usage

```
pmgetSignature(mutationFeatureData, K, numInit = 10, tol = 1e-04,
              maxIter = 10000)
```

Arguments

mutationFeatureData	the mutation data (MutationFeatureData class (S4 class)) by the hildaReadMPFile.
K	the number of mutation signatures
numInit	the number of performing calculations with different initial values
tol	tolerance for the estimation (when the difference of log-likelihoods become below this value, stop the estimation)
maxIter	the maximum number of iteration of estimation

Value

The output is an instance of EstimatedParameters S4 class, which stores estimated parameters and other meta-information, and will be used for saving parameter values and visualizing the mutation signatures and memberships

Examples

```
## After obtaining G (see e.g., hildaReadMPFile function)
load(system.file("extdata/sample.rdata", package="HiLDA"))
Param <- pmgetSignature(G, K = 3)
```

pmMultiBarplot	<i>Plot both mutation signatures and their mutational exposures from pm-signature output for more than two groups</i>
----------------	-----------------------------------------------------------------------------------------------------------------------

Description

Plot both mutation signatures and their mutational exposures from pmsignature output for more than two groups

Usage

```
pmMultiBarplot(inputG, inputParam, sigOrder = NULL, groupIndices,
              sortSampleNum = TRUE, charSize = 3)
```


Arguments

inputG	a MutationFeatureData S4 class output by the pmsignature.
inputParam	a estimatedParameters S4 class output by the pmsignature.
sigOrder	the order of signatures if needed (default: NULL).
groupIndices	a vector of group indicators.
sortSampleNum	an indicator variable on whether samples are sorted by the number of mutations (default: TRUE).
charSize	the size of the character on the signature plot (default: 3)

Value

a list of the signature plot and the mean difference plot.

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
Param <- pmgetSignature(G, K = 3)

pmPlots <- pmMultiBarplot(G, Param, groupIndices=c(1, rep(2,3), rep(3,6)))
cowplot::plot_grid(pmPlots$sigPlot, pmPlots$propPlot, rel_widths = c(1,3))
```

pmPlotSignature	<i>Plot mutation signatures from pmsignature output</i>
-----------------	---------------------------------------------------------

Description

Plot mutation signatures from pmsignature output

Usage

```
pmPlotSignature(inputParam, sigOrder = NULL, colorList = NULL, ...)
```

Arguments

inputParam	a estimatedParameters S4 class output by the pmsignature.
sigOrder	the order of signatures if needed (default: NULL).
colorList	a list of color to highlight the signatures (default: NULL).
...	additional arguments passed on to visPMS.

Value

a plot object containing all mutational signatures

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
Param <- pmgetSignature(G, K = 3)
pmPlotSignature(Param)
```

PMSboundary	<i>A functional for generating the function checking the parameter (p) is within the restricted conditions or not</i>
-------------	-----------------------------------------------------------------------------------------------------------------------

Description

A functional for generating the function checking the parameter (p) is within the restricted conditions or not

Usage

PMSboundary(y)

Arguments

y	this variable includes the information on the mutation features, the number of mutation signatures specified and so on
---	------------------------------------------------------------------------------------------------------------------------

Value

a functional

updateMstepFQC	<i>Update the parameter F and Q (M-step in the EM-algorithm)</i>
----------------	------------------------------------------------------------------

Description

Update the parameter F and Q (M-step in the EM-algorithm)

Usage

```
updateMstepFQC(vPatternList, vSparseCount, nTheta, fdim, signatureNum,
  sampleNum, patternNum, samplePatternNum, isBackground)
```

Arguments

vPatternList	The list of possible mutation features (converted to a vector)
vSparseCount	The table showing (mutation feature, sample, the number of mutation) (converted to a vector)
nTheta	The parameters in the distribution
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes
patternNum	the number of possible combinations of all the mutation features
samplePatternNum	the number of possible combination of samples and mutation patterns
isBackground	the logical value showing whether a background mutation features is included or not

Value

a vector

updatePMSPParam	<i>A function for updating parameters using EM-algorithm</i>
-----------------	--------------------------------------------------------------

Description

A function for updating parameters using EM-algorithm

Usage

```
updatePMSPParam(p, y)
```

Arguments

p	this variable includes the parameters for mutation signatures and membership parameters
y	this variable includes the information on the mutation features, the number of mutation signatures specified and so on

Value

a value

updateTheta_NormalizedC	<i>Update the auxiliary parameters theta and normalize them so that the summation of each group sums to 1 (E-step), also calculate the current log-likelihood value</i>
-------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Description

Update the auxiliary parameters theta and normalize them so that the summation of each group sums to 1 (E-step), also calculate the current log-likelihood value

Usage

```
updateTheta_NormalizedC(vPatternList, vSparseCount, vF, vQ, fdim,
signatureNum, sampleNum, patternNum, samplePatternNum, isBackground, vF0)
```

Arguments

vPatternList	The list of possible mutation features (converted to a vector)
vSparseCount	The table showing (mutation feature, sample, the number of mutation) (converted to a vector)
vF	F (converted to a vector)
vQ	Q (converted to a vector)
fdim	a vector specifying the number of possible values for each mutation signature
signatureNum	the number of mutation signatures
sampleNum	the number of cancer genomes
patternNum	the number of possible combinations of all the mutation features
samplePatternNum	the number of possible combination of samples and mutation patterns
isBackground	the logical value showing whether a background mutation features is included or not
vF0	a background mutation features

Value

a value for theta

visPMS	<i>visualize probabilistic mutation signature for the independent model</i>
--------	-----------------------------------------------------------------------------

Description

Generate visualization of mutation signatures for the model with substitution patterns and flanking bases represented by the independent representation.

Usage

```
visPMS(vF, numBases, baseCol = NA, trDir = FALSE, charSize = 5,
       isScale = FALSE, alpha = 2, charLimit = 0.25)
```

Arguments

vF	a matrix for mutation signature
numBases	the number of flanking bases
baseCol	the colour of the bases (A, C, G, T, plus/minus strand)
trDir	the index whether the strand direction is plotted or not
charSize	the size of the character
isScale	the index whether the height of the flanking base is changed or not
alpha	the parameter for the Renyi entropy (applicable only if the isScale is TRUE)
charLimit	the limit of char size

Value

a plot of the input mutational signature

Examples

```
load(system.file("extdata/sample.rdata", package="HiLDA"))
Param <- pmgetSignature(G, K = 3)

sig <- slot(Param, "signatureFeatureDistribution")[1,]
visPMS(sig, numBases = 5, isScale = TRUE)
```

Index

boundaryTurbo_F, [2](#)
boundaryTurbo_Q, [3](#)

calcPMSLikelihood, [3](#)
convertFromTurbo_F, [4](#)
convertFromTurbo_Q, [4](#)
convertToTurbo_F, [5](#)
convertToTurbo_Q, [5](#)

EstimatedParameters-class, [6](#)

getLogLikelihoodC, [6](#)
getMutationFeatureVector, [7](#)

hildaBarplot, [7](#)
hildaDiffPlot, [8](#)
hildaGlobalResult, [9](#)
hildaLocalResult, [9](#)
hildaPlotSignature, [10](#)
hildaReadMPFile, [11](#)
hildaRhat, [12](#)
hildaTest, [12](#)

MetaInformation-class, [13](#)
MutationFeatureData-class, [14](#)
mySquareEM, [14](#)

pmBarplot, [15](#)
pmgetSignature, [16](#)
pmMultiBarplot, [16](#)
pmPlotSignature, [17](#)
PMSboundary, [18](#)

updateMstepFQC, [18](#)
updatePMSPParam, [19](#)
updateTheta_NormalizedC, [19](#)

visPMS, [20](#)