Package ‘motifTestR’

July 17, 2025

Title Perform key tests for binding motifs in sequence data

Version 1.5.0

Description Taking a set of sequence motifs as PWMs, test a set of sequences for over-representation of these motifs, as well as any positional features within the set of motifs. Enrichment analysis can be undertaken using multiple statistical approaches. The package also contains core functions to prepare data for analysis, and to visualise results.

License GPL-3

Encoding UTF-8

URL `https://github.com/smped/motifTestR`

BugReports `https://github.com/smped/motifTestR/issues`

Depends Biostrings, GenomicRanges, ggplot2 (>= 3.5.0), R (>= 4.3.0),

Imports GenomeInfoDb, graphics, harmonicmeanp, IRanges, matrixStats, methods, parallel, patchwork, rlang, S4Vectors, stats, universalmotif,

Suggests AnnotationHub, BiocStyle, BSgenome.Hsapiens.UCSC.hg19, ComplexUpset, extraChIPs, ggdendro, knitr, MASS, MotifDb, rmarkdown, rtracklayer, testthat (>= 3.0.0), VGAM

biocViews MotifAnnotation, ChIPSeq, ChipOnChip, SequenceMatching, Software

LazyData false

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

VignetteBuilder knitr

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

git_branch devel

git_last_commit bdc1119

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-07-16

Author Stevie Pederson [aut, cre] (ORCID: <`https://orcid.org/0000-0001-8197-3303`>)

Maintainer Stevie Pederson `<stephen.pederson.au@gmail.com>`

Contents

`motifTestR-package` *motifTestR: Perform Key Analyses on Transcription Factor Binding Motifs*

Description

The package `motifTestR` has been designed for two primary analyses of TFBMs, testing for positional bias and overall enrichment.

Details

The package `motifTestR` provides two primary functions for testing TFBMs within a set of sequences

- `testMotifPos()` for detecting positional bias within a set of test sequences
- `testMotifEnrich()` for testing overall enrichment of a TFBM within a set of test sequences

Motifs are also able to be clustered for analysis as a cluster, or for grouping results. Clusters from external approaches can also be incorporated.

- `testClusterPos()` for detecting positional bias for matches to any motif annotated to a cluster, within a set of test sequences
- `testClusterEnrich()` for testing overall enrichment of any TFBM annotated to a cluster, within a set of test sequences

The main functions rely on lower-level functions such as:

- `countPwmMatches()` simply counts the number of matches within an `XStringSet`
- `getPwmMatches()` returns the position of matches within an `XStringSet`
- `countClusterMatches()` simply counts the number of matches to motifs annotated to a cluster within an `XStringSet`

- `getClusterMatches()` returns the position of matches to motifs annotated to a cluster within an `XStringSet`
- `makeRMRanges()` which produces a set of random, matching ranges based on key characteristics of the set of test sequences/ranges

A simple utility function is provided to enable visualisation of results

- `plotMatchPos()` enables visualisation of the matches within a set of sequences using multiple strategies

Author(s)

Stevie Pederson

See Also

Useful links:

- `https://github.com/smped/motifTestR`
- Report bugs at `https://github.com/smped/motifTestR/issues`

`ar_er_peaks` *A set of peaks with AR and ER detected*

Description

A set of ChIP-Seq peaks where AR and ER were both detected

Usage

```
data("ar_er_peaks")
```

Format

An object of class `GRanges` of length 849.

Details

The subset of peaks found on chr1 which contained signal from at least two of AR, ER and H3K27ac, taken from GSE123767. Peaks were resized to a uniform width of 400bp after downloading

Generation of these ranges is documented in `system.file("scripts/ar_er_peaks.R", package = "motifTestR")`

Source

`https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE123767`

Examples

```
data("ar_er_peaks")
ar_er_peaks
```

`ar_er_seq` *Sequences from peaks with AR and ER detected*

Description

The genomic sequences obtained from the ar_er_peaks

Usage

```
data("ar_er_seq")
```

Format

An object of class `DNAStringSet` of length 849.

Details

These sequences represent the sequences obtained from BSgenome.Hsapiens.UCSC.hg19 for thw peaks supplied as `ar_er_peaks`

Generation of these sequences is documented in `system.file("scripts/ar_er_peaks.R", package = "motifTestR")`

Examples

```
data("ar_er_seq")
ar_er_seq
```

`clusterMotifs` *Assign each motif to a cluster*

Description

Cluster related motifs for testing as a group

Usage

```
clusterMotifs(
  motifs,
  type = c("PPM", "ICM"),
  method = c("PCC", "EUCL", "SW", "KL", "ALLR", "BHAT", "HELL", "SEUCL", "MAN",
    "ALLR_LL", "WEUCL", "WPCC"),
  power = 1,
  agglom = "complete",
  thresh = 0.2,
  return_d = FALSE,
  plot = FALSE,
  labels = FALSE,
  cex = 1,
  linecol = "red",
  ...
)
```

Arguments

`motifs`	A list of universalmotifs or a list of PWMs
`type`	Can be ICM or PPM
`method`	The method to be used for determining similarity/distances
`power`	Raise correlation matrices to this power before converting to a distance matrix. Only applied if method is either "PCC" or "WPCC"
`agglom`	Method to be used for agglomeration by hclust
`thresh`	Tree heights below which motifs are formed into a cluster
`return_d`	logical(1) Return the distance matrices for each cluster
`plot`	Show tree produced by hclust. If requested the value set by thresh will be shown as a horizontal line
`labels, cex`	Passed to plot.hclust
`linecol`	Passed to abline as the argument `col`
`...`	passed to compare_motifs

Details

This builds on compare_motifs, enabling the assignment of each PWM to a cluster, and subsequent testing of motifs as a cluster, rather than returning individual results.

Internally all matrices are converted to distance matrices and hclust is used to form clusters. By default, options such as "EUCL", "MAN" produce distances, whilst similarity matrices are produced when choosing "PCC" and other correlation based metrics. In these cases, the distance matrix is obtained by taking 1 - similarity.

By default PWM labels are hidden (labels = FALSE), however these can be shown using labels = NULL as explained in plot.hclust.

Raising the threshold will lead to fewer, larger clusters whilst leaving this value low will return a more conservative approach, with more smaller clusters. The final decision as the best clustering strategy is highly subjective and left to the user. Manual inspection of motifs within a cluster can be performed using view_motifs, as shown in the vignette.

Value

Named vector with numeric values representing which cluster each motif has been assigned to.

If setting return_d = TRUE, a named list will be returned with the clusters as the element `cl` and a list with distance matrices for each cluster as the element `d`

Examples

```
# Load the example motifs
data("ex_pfm")

# Return a vector with each motif assigned a cluster
# The default uses Pearson's Correlation Coefficient
clusterMotifs(ex_pfm)

# Preview the settings noting that showing labels can clutter the plot
# with large numbers of motifs. The defaults for Euclidean distance
# show the threshold may need raising
clusterMotifs(ex_pfm, plot = TRUE, labels = NULL, method = "EUCL")
```

`countPwmMatches` *Count the matches to a PWM within an XStringSet*

Description

Count the matches to a PWM within an XStringSet

Usage

```
countPwmMatches(
  pwm,
  stringset,
  rc = TRUE,
  min_score = "80%",
  mc.cores = 1,
  ...
)
```

Arguments

`pwm`	A Position Weight Matrix
`stringset`	An XStringSet
`rc`	logical(1) Also find matches using the reverse complement of pwm
`min_score`	The minimum score to return a match
`mc.cores`	Passed to mclapply when analysing a list of PWMs
`...`	Passed to countPWM

Details

Will simply count the matches within an XStringSet and return an integer. All matches are included.

Value

An integer vector

Examples

```
## Load the example PWM
data("ex_pfm")
esr1 <- ex_pfm$ESR1

## Load the example Peaks
data("ar_er_seq")
countPwmMatches(esr1, ar_er_seq)

## Count all PWMs
countPwmMatches(ex_pfm, ar_er_seq)
```

`ex_pfm` *Example Position Frequency Matrices*

Description

Example Position Frequency Matrices

Usage

```
data("ex_pfm")
```

Format

An object of class `list` of length 5.

Details

This object contains 5 PFMs taken from HOCOMOCOv11-coreA for examples and testing

Generation of this motif list is documented in `system.file("scripts/ex_pfm.R", package = "motifTestR")`

Examples

```
data("ex_pfm")
ex_pfm$ESR1
```

`getClusterMatches` *Find matches from a PWM cluster within an XStringSet*

Description

Find matches from a PWM cluster within a set of sequences

Usage

```
getClusterMatches(
  cl,
  stringset,
  rc = TRUE,
  min_score = "80%",
  best_only = FALSE,
  break_ties = c("all", "random", "first", "last", "central"),
  mc.cores = 1,
  ...
)

countClusterMatches(
  cl,
  stringset,
```

```
  rc = TRUE,
  min_score = "80%",
  mc.cores = 1,
  ...
)
```

Arguments

`cl`	A list of Position Weight Matrices, universalmotifs, with each element representing clusters of related matrices
`stringset`	An XStringSet
`rc`	logical(1) Also find matches using the reverse complement of PWMs in the cluster
`min_score`	The minimum score to return a match
`best_only`	logical(1) Only return the best match
`break_ties`	Method for breaking ties when only returning the best match Ignored when all matches are returned (the default)
`mc.cores`	Passed to mclapply
`...`	Passed to matchPWM

Details

This function extends getPwmMatches by returning a single set of results for set of clustered motifs. This can help remove some of the redundancy in results returned for highly similar PWMs, such as those in the GATA3 family.

Taking a set of sequences as an XStringSet, find all matches above the supplied score (i.e. threshold) for a list of Position Weight Matrices (PWMs), which have been clustered together as highly-related motifs. By default, matches are performed using the PWMs as provided and the reverse complement, however this can easily be disabled by setting `rc = FALSE`.

The function relies heavily on matchPWM and Views for speed.

Where overlapping matches are found for the PWMs within a cluster, only a single match is returned. The motif with the highest relative score (score / maxScore(PWM)) is selected.

When choosing to return the best match (`best_only = TRUE`), only the match with the highest relative score is returned for each sequence. Should there be tied scores, the best match can be chosen as either the first, last, most central, all tied matches, or choosing one at random (the default).

Value

Output from getClusterMatches will be a list of DataFrames with columns: `seq`, `score`, `direction`, `start`, `end`, `from_centre`, `seq_width`, `motif` and `match`

The first three columns describe the sequence with matches, the score of the match and whether the match was found using the forward or reverse PWM. The columns `start`, `end` and `width` describe the where the match was found in the sequence, whilst `from_centre` defines the distance between the centre of the match and the centre of the sequence being queried. The motif column denotes which individual motif was found to match in this position, again noting that when matches overlap, only the one with the highest relative score is returned. The final column contains the matching fragment of the sequence as an `XStringSet`.

Output from countClusterMatches will be a simple integer vector the same length as the number of clusters

Examples

```
# Load example PFMs
data("ex_pfm")
# Cluster using default settings
cl_ids <- clusterMotifs(ex_pfm)
ex_cl <- split(ex_pfm, cl_ids)
# Add optional names
names(ex_cl) <- vapply(ex_cl, \(x) paste(names(x), collapse = "/"), character(1))

# Load example sequences
data("ar_er_seq")
# Get all matches for each cluster
getClusterMatches(ex_cl, ar_er_seq)
# Or Just count them
countClusterMatches(ex_cl, ar_er_seq)
# Compare this to individual counts
countPwmMatches(ex_pfm, ar_er_seq)
```

`getPwmMatches` *Find all PWM matches within an XStringSet*

Description

Find all PWM matches within a set of sequences

Usage

```
getPwmMatches(
  pwm,
  stringset,
  rc = TRUE,
  min_score = "80%",
  best_only = FALSE,
  break_ties = c("all", "random", "first", "last", "central"),
  mc.cores = 1,
  ...
)
```

Arguments

`pwm`	A Position Weight Matrix, list of PWMs or universalmotif list
`stringset`	An XStringSet
`rc`	logical(1) Also find matches using the reverse complement of pwm
`min_score`	The minimum score to return a match
`best_only`	logical(1) Only return the best match
`break_ties`	Method for breaking ties when only returning the best match Ignored when all matches are returned (the default)
`mc.cores`	Passed to mclapply if passing multiple PWMs
`...`	Passed to matchPWM

Details

Taking a set of sequences as an XStringSet, find all matches above the supplied score (i.e. threshold) for a single Position Weight Matrix (PWM), generally representing a transcription factor binding motif. By default, matches are performed using the PWM as provided and the reverse complement, however this can easily be disabled by setting `rc = FALSE`.

The function relies heavily on matchPWM and Views for speed.

When choosing to return the best match (`best_only = TRUE`), only the match with the highest score is returned for each sequence. Should there be tied scores, the best match can be chosen as either the first, last, most central, all tied matches, or choosing one at random (the default).

Value

A DataFrame with columns: `seq`, `score`, `direction`, `start`, `end`, `from_centre`, `seq_width`, and `match`

The first three columns describe the sequence with matches, the score of the match and whether the match was found using the forward or reverse PWM. The columns `start`, `end` and `width` describe the where the match was found in the sequence, whilst `from_centre` defines the distance between the centre of the match and the centre of the sequence being queried. The final column contains the matching fragment of the sequence as an `XStringSet`.

When passing a list of PWMs, a list of the above DataFrames will be returned.

Examples

```
## Load the example PWM
data("ex_pfm")
esr1 <- ex_pfm$ESR1

## Load the example Peaks
data("ar_er_seq")

## Return all matches
getPwmMatches(esr1, ar_er_seq)

## Just the best match
getPwmMatches(esr1, ar_er_seq, best_only = TRUE)

## Apply multiple PWMs as a list
getPwmMatches(ex_pfm, ar_er_seq, best_only = TRUE)
```

`hg19_mask` *Regions from hg19 with high N content*

Description

A GRanges object with regions annotated as telomeres or centromeres

Usage

```
data("hg19_mask")
```

Format

An object of class `GRanges` of length 345.

Details

The regions defined as centromeres or telomeres in hg19, taken from AnnotationHub objects "AH107360" and "AH107361". These were combined with regions containing Ns from the UCSC 2bit file, and regions with Ns in the BSgenome.Hsapiens.UCSC.hg19 were retained.

Generation of these ranges is documented in `system.file("scripts/hg19_mask.R", package = "motifTestR")`

Source

The package AnnotationHub and https://hgdownload.cse.ucsc.edu/goldenpath/hg19/bigZips/hg19.fa.masked.gz

Examples

```
data("hg19_mask")
hg19_mask
```

---

makeRMRanges *Form a set of random, matching ranges for bootstrapping or permuting*

Description

Form a set of ranges from y which (near) exactly match those in x for use as a background set requiring matching

Usage

```
makeRMRanges(x, y, ...)

## S4 method for signature 'GRanges,GRanges'
makeRMRanges(
  x,
  y,
  exclude = GRanges(),
  n_iter = 1,
  n_total = NULL,
  replace = TRUE,
  ...,
  force_ol = TRUE
)

## S4 method for signature 'GRangesList,GRangesList'
makeRMRanges(
  x,
  y,
  exclude = GRanges(),
```

```
  n_iter = 1,
  n_total = NULL,
  replace = TRUE,
  mc.cores = 1,
  ...,
  force_ol = TRUE,
  unlist = TRUE
)
```

Arguments

`x`	GRanges/GRangesList with ranges to be matched
`y`	GRanges/GRangesList with ranges to select random matching ranges from
`...`	Not used
`exclude`	GRanges of ranges to omit from testing
`n_iter`	The number of times to repeat the random selection process
`n_total`	Setting this value will over-ride anything set using n_iter. Can be vector of any length, corresponding to the length of x, when x is a GRangesList
`replace`	logical(1) Sample with our without replacement when creating the set of random ranges.
`force_ol`	logical(1) Enforce an overlap between every site in x and y
`mc.cores`	Passsed to mclapply
`unlist`	logical(1) Return as a sorted GRanges object, or leave as a GRangesList

Details

This function uses the width distribution of the ’test’ ranges (i.e. `x`) to randomly sample a set of ranges with matching width from the ranges provided in `y`. The width distribution will clearly be exact when a set of fixed-width ranges is passed to `x`, whilst random sampling may yield some variability when matching ranges of variable width.

When both `x` and `y` are GRanges objects, they are implicitly assumed to both represent similar ranges, such as those overlapping a promoter or enhancer. When passing two GRangesList objects, both objects are expected to contain ranges annotated as belonging to key features, such that the list elements in `y` must encompass all elements in `x`. For example if `x` contains two elements named ’promoter’ and ’intron’, `y` should also contain elements named ’promoter’ and ’intron’ and these will be sampled as matching ranges for the same element in `x`. If elements of `x` and `y` are not named, they are assumed to be in matching order.

The default behaviour is to assume that randomly-generated ranges are for iteration, and as such, ranges are randomly formed in multiples of the number of ’test’ ranges provided in `x`. The column `iteration` will be added to the returned ranges. Placing any number into the `n_total` argument will instead select a total number of ranges as specified here. In this case, no `iteration` column will be included in the returned ranges.

Sampling is assumed to be with replacement as this is most suitable for bootstrapping and related procedures, although this can be disabled by setting `replace = FALSE`

Value

A GRanges or GRangesList object

Examples

```
## Load the example peaks
data("ar_er_peaks")
sq <- seqinfo(ar_er_peaks)
## Now sample size-matched ranges for two iterations from chr1
makeRMRanges(ar_er_peaks, GRanges(sq)[1], n_iter = 2)

## Or simply sample 100 ranges if not planning any iterative analyses
makeRMRanges(ar_er_peaks, GRanges(sq)[1], n_total = 100)
```

`plotMatchPos` *Plot Motif Match Positions*

Description

Plot the distribution of motif matches across sequences

Usage

```
plotMatchPos(
  matches,
  binwidth = 10,
  abs = FALSE,
  use_totals = FALSE,
  type = c("density", "cdf", "heatmap"),
  geom = c("smooth", "line", "point", "col"),
  cluster = FALSE,
  w = 0.1,
  heat_fill = NULL,
  ...
)
```

Arguments

`matches`	Output from getPwmMatches
`binwidth`	Width of bins to use when plotting
`abs`	logical(1) Plot absolute distances from centre
`use_totals`	logical(1). If TRUE, plots will use total counts. The default (FALSE) plots probabilities.
`type`	Plot match density, the CDF or a binned heatmap
`geom`	Type of geom to be used for line plots. Ignored for heatmaps
`cluster`	logical(1) Cluster motifs when drawing a heatmap. If TRUE a dendrogram will be added to the LHS of the plot
`w`	Relative width of the dendrogram on (0, 1)
`heat_fill`	scale_fill_continuous object for heatmaps. If not provided, scale_fill_viridis_c() will be added to the heatmap.
`...`	Passed to individual geom* functions

Details

Multiple options are provided for showing the distribution of PWM matches within a set of sequences, using either the smoothed probability density, the probability CDF or as a heatmap. Distances can be shown as symmetrical around the centre or using absolute distances from the central position within the sequences.

Heatmaps are only enabled for comparisons across multiple PWMs, with optional clustering enabled. If adding a dendrogram for clustering, the returned plot object will be a patchwork object.

Value

A ggplot2 object

Examples

```
## Load the example PWM
data("ex_pfm")
esr1 <- ex_pfm$ESR1

## Load the example sequences from the peaks
data("ar_er_seq")

## Just the best match
bm <- getPwmMatches(esr1, ar_er_seq, best_only = TRUE)
plotMatchPos(bm, se = FALSE)

## Matches can also be shown by distance from centre
plotMatchPos(bm, abs = TRUE)

## Cumulative Probability plots are also implemented
plotMatchPos(bm, type = "cdf", geom = "line", colour = "red") +
  geom_abline(intercept = 0.5, slope = 1/ 400)
```

`simSeq` *Simulate sequences using optional TFBMs*

Description

Simulate a set of fixed-width sequences using optional TFBMs

Usage

```
simSeq(
  n,
  width,
  pfm = NULL,
  nt = c("A", "C", "G", "T"),
  prob = rep(0.25, 4),
  shape1 = 1,
  shape2 = 1,
  rate = NULL,
  theta = NULL,
```

```
  as = "DNAStringSet",
  ...
)
```

Arguments

`n`	The number of sequences to simulate
`width`	Width of sequences to simulate
`pfm`	Probability Weight/Frequency Matrix
`nt`	Nucleotides to include
`prob`	Sampling probabilities for each nucleotide
`shape1, shape2`	Passed to rbetabinom.ab
`rate`	The expected rate of motifs per sequence. Is equivalent to λ in rpois. If set to NULL, all sequences will be simulated with a single motif, otherwise a Poisson distribution will be used
`theta`	Overdispersion parameter passed to rnegbin. If set to NULL the rate parameter will be passed to rpois. However if this value is set, the rate and theta parameters are passed to rnegbin to simulate overdispersed counts
`as`	ObjectClass to return objects as. Defaults to DNAStringSet, but other viable options may include 'character', 'CharacterList' or any other class from which a character vector may be coerced.
`...`	Not used

Details

Using the nucleotide and probabilities provided as set of sequences can be simulated. By default, this will effectively be a set of 'background' sequences, with letters effectively chosen at random.

If a PWM/PFM is supplied, the shape parameters are first passed to rbetabinom.ab to determine the random positions the motif will be placed, with the default parameters representing a discrete uniform distribution.

The sequences to have a motif inserted will be selected, along with the number of motifs, using the rate and theta parameters. If both are NULL, every sequence will have a single motif inserted. If the rate is > 0 and theta is NULL, sequences will be selected to have motifs inserted using a poisson distribution. If theta is also provided, sequences will be selected to contain motifs using a negative binomial distribution

Once positions and sequences for the TFBM have been selected, nucleotides will be randomly sampled using the probabilities provided in the PWM and these motifs will be placed at the randomly sampled positions

Value

By default a DNAStringSet will be returned. If possible, the position of any randomly sampled motifs will be included in the mcols element of the returned object.

Examples

```
## Randomly generate 10x50nt sequences without any TFBMs present
simSeq(10, 50)

## Now place a motif at random positions
```

```
data('ex_pfm')
sim_seq <- simSeq(10, width = 20, pfm = ex_pfm$ESR1)
sim_seq
## The position of the motif within each sequence is included in the mcols
mcols(sim_seq)

## Use this to extract the random motifs from the random sequences
library(IRanges)
i <- mcols(sim_seq)$pos + cumsum(width(sim_seq)) - width(sim_seq)
Views(unlist(sim_seq), start = i, width = 10)
```

`testClusterEnrich` *Test enrichment across a cluster of motifs using a background set of sequences*

Description

Test for enrichment of any motif within a cluster across a set of sequences using a background set to derive a NULL hypothesis

Usage

```
testClusterEnrich(
  cl,
  stringset,
  bg,
  var = "iteration",
  model = c("quasipoisson", "hypergeometric", "poisson", "iteration"),
  sort_by = c("p", "none"),
  mc.cores = 1,
  prior.count = 1,
  seed = 100,
  ...
)
```

Arguments

`cl`	A list of Position Weight Matrices, universalmotifs, with each element representing clusters of related matrices
`stringset`	An XStringSet with equal sequence widths
`bg`	An XStringSet with the same sequence widths as the test XStringset
`var`	A column in the mcols element of bg, usually denoting an iteration number
`model`	The model used for analysis
`sort_by`	Column to sort results by
`mc.cores`	Passed to mclapply
`prior.count`	Added to all counts to better manage zero counts in background sequences. For analysis under QuasiPoisson models prior counts are added as Poisson noise using this value as expected counts
`seed`	Used for reproducibility when adding Poisson noise
`...`	Passed to getPwmMatches or countPwmMatches

Details

This extends the analytic methods offered by testMotifEnrich using PWMs grouped into a set of clusters. As with all cluster-level approaches, hits from multiple PWMs which overlap are counted as a single hit ensuring that duplicated matches are not double-counted, and that only individual positions within the sequences are.

Value

See testMotifEnrich

See Also

`makeRMRanges()`, `getClusterMatches()`, `countClusterMatches()`, `testMotifEnrich()`

Examples

```
## Load the example peaks & the sequences
data("ar_er_peaks")
data("ar_er_seq")
sq <- seqinfo(ar_er_peaks)
## Now sample size-matched ranges 10 times larger. In real-world analyses,
## this set should be sampled as at least 1000x larger, ensuring features
## are matched to your requirements. This example masks regions with known N
## content, including centromeres & telomeres
data("hg19_mask")
set.seed(305)
bg_ranges <- makeRMRanges(
  ar_er_peaks, GRanges(sq)[1], exclude = hg19_mask, n_iter = 10
)

## Convert ranges to DNAStringSets
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19
bg_seq <- getSeq(genome, bg_ranges)

## Test for enrichment of clustered motifs
data("ex_pfm")
cl <- list(A = ex_pfm[1], B = ex_pfm[2:3])
testClusterEnrich(cl, ar_er_seq, bg_seq, model = "poisson")
```

`testClusterPos` *Test positional bias motifs within a cluster*

Description

Test positional bias for all motifs within a given cluster

Usage

```
testClusterPos(
  x,
  stringset,
  binwidth = 10,
  abs = FALSE,
  rc = TRUE,
  min_score = "80%",
  break_ties = "all",
  alt = c("greater", "less", "two.sided"),
  sort_by = c("p", "none"),
  mc.cores = 1,
  ...
)
```

Arguments

`x`	A Position Weight Matrix, universalmotif object or list thereof. Alternatively can be a single DataFrame or list of DataFrames as returned by getClusterMatches with `best_only = TRUE`
`stringset`	An XStringSet. Not required if matches are supplied as x
`binwidth`	Width of bins across the range to group data into
`abs`	Use absolute positions around zero to find symmetrical enrichment
`rc`	logical(1) Also find matches using the reverse complement of each PWM
`min_score`	The minimum score to return a match
`break_ties`	Choose how to resolve matches with tied scores
`alt`	Alternative hypothesis for the binomial test
`sort_by`	Column to sort results by
`mc.cores`	Passed to mclapply
`...`	Passed to matchPWM

Details

This is a reimplementation of testMotifPos for sets of motifs which have been clustered for similarity. The positions test the bias of any motifs within the cluster given that overlapping matches are only counted once, and with the match retained being the one with the highest relative score.

It should also be noted that some motif clusters will contain PWMs of varying length. When finding positional bias, the widest motif is taken as the width for all, and any matches from narrower motifs outside of the range allowed by wider motifs are discarded. This reduction in signal will make a small difference in the outer bins, but is not considered to be problematic for the larger analysis.

Value

A data.frame with columns `start`, `end`, `centre`, `width`, `total_matches`, `matches_in_region`, `expected`, `enrichment`, `prop_total`, `p` and `consensus_motif` The total matches represent the total number of matches within the set of sequences, whilst the number observed in the final region are also given, along with the proportion of the total this represents. Enrichment is simply the ratio of observed to expected based on the expectation of the null hypothesis

The consensus motif across all matches is returned as a Position Frequency Matrix (PFM) using consensusMatrix.

Examples

```
## Load the example PWM
data("ex_pfm")
## Load the example sequences
data("ar_er_seq")

## Cluster the motifs
cl <- list(A = ex_pfm[1], B = ex_pfm[2:3])

## Get the best match and use this data
matches <- getClusterMatches(cl, ar_er_seq, best_only = TRUE)
## Test for enrichment in any position
testClusterPos(matches)

## Or just pass the clustered matrices
## Here we've set abs = TRUE to test absolute distance from the centre
testClusterPos(cl, ar_er_seq, abs = TRUE, binwidth = 10)
```

`testMotifEnrich` *Test motif enrichment using a background set of sequences*

Description

Test for motif enrichment within a set of sequences using a background set to derive a NULL hypothesis

Usage

```
testMotifEnrich(
  pwm,
  stringset,
  bg,
  var = "iteration",
  model = c("quasipoisson", "hypergeometric", "poisson", "iteration"),
  sort_by = c("p", "none"),
  mc.cores = 1,
  prior.count = 1,
  seed = 100,
  ...
)
```

Arguments

`pwm`	A Position Weight Matrix or list of PWMs
`stringset`	An XStringSet with equal sequence widths
`bg`	An XStringSet with the same sequence widths as the test XStringset
`var`	A column in the mcols element of bg, usually denoting an iteration number
`model`	The model used for analysis
`sort_by`	Column to sort results by

`mc.cores`	Passed to mclapply
`prior.count`	Added to all counts to better manage zero counts in background sequences. For analysis under QuasiPoisson models prior counts are added as Poisson noise using this value as expected counts
`seed`	Used for reproducibility when adding Poisson noise
`...`	Passed to getPwmMatches or countPwmMatches

Details

This function offers four alternative models for assessing the enrichment of a motif within a set of sequences, in comparison to a background set of sequences. Selection of the BG sequences plays an important role and, in conjunction with the question being addressed, determines the most appropriate model to use for testing, as described below.

It should also be noted that the larger the BG set of sequences, the larger the computational burden, and results can take far longer to return. For many millions of background sequences, this may run beyond an hour

Descriptions of Models and Use Cases:

Hypergeometric Tests:
Hypergeometric tests are best suited to the use case where the test set of sequences represents a subset of a larger set, with a specific feature or behaviour, whilst the BG set may be the remainder of the set without that feature. For example, the test set may represent ChIP-Seq binding sites where signal changes in response to treatment, whilst the BG set represents the sites where no changed signal was observed. Testing is one-sided, for enrichment of motifs within the test set.
Due to these relatively smaller sized datasets, setting model = "hypergeometric", will generally return results quickly

Poisson Tests:
This approach requires a set of background sequences which should be much larger than the test set of sequences. The parameters for a Poisson model are estimated in a per-sequence manner on the set of BG sequences, and the observed rate of motif-matches within the test set is then tested using poisson.test. Testing is two-sided.
This approach assumes that all matches follow a Poisson distribution, which is often true, but data can also be over-dispersed. Given that this model can also return results relatively quickly, is it primarily suitable for data exploration, such as quickly checking for expected behaviours, but not for final results.

Quasi-Poisson Test:
The quasipoisson model allows for over-dispersion and will return more conservative results than using the standard Poisson model. Under the method currently implemented here, BG sequences should be divided into blocks (i.e. iterations), identical in size to the test set of sequences. Model parameters are estimated per iteration across the BG set of sequences, with the rate of matches in the test set being compared against these blocks. This ensures more conservative results that if analysing test and bg sequences as collections of individual sequences.
It is expected that the BG set will matched for the features of interest and chosen using makeRMRanges with a large number of iterations, e.g. n_iter = 1000. Due to this parameterisation, quasipoisson approaches can be computationally time-consuming, as this is effectively an iterative approach. Testing is two-sided.

Iteration:
Setting the model as "iteration" performs a non-parametric analysis, with the exception of returning Z-scores under the Central Limit Theorem. Mean and SD of matches is found for each

iteration, and used to return Z scores, with p-values returned from both a Z-test and from comparing observed values directly to sampled values obtained from the BG sequences. Sampled values are calculated directly and as such, are limited in precision.
As for the QuasiPoisson model, a very large number of iterations is expected to be used, to ensure the CLT holds, again making this a computationally demanding test. Each iteration/block is expected to be identically-sized to the test set, and matched for any features as appropriate using `makeRMRanges()`.

Value

A data.frame with columns: `sequences`, `matches`, `expected`, `enrichment`, and `p`, with additional columns `Z`, `est_bg_rate` (Poisson), `odds_ratio` (Hypergeometric) or `Z`, `sd_bg`, `n_iter` and `iter_p` (Iterations). The numbers of sequences and matches refer to the test set of sequences, whilst expected is the expected number of matches under the Poisson or iterative null distribution. The ratio of matches to expected is given as `enrichment`, along with the Z score and p-value. Whilst the Z score is only representative under the Poisson model, it is used to directly estimate p-values under the iterative approach. Under this latter approach, the sd of the matches found in the background sequences is also given, along with the number of iterations and the p-values from permutations testing the one-sided hypothesis hypothesis for enrichment.

It may also be worth noting that if producing background sequences using makeRMRanges with `replace = TRUE` and `force_ol = TRUE`, the iterative model corresponds to a bootstrap, given that the test sequences will overlap the background sequences and background ranges are able to be sampled with replacement.

See Also

`makeRMRanges()`, `getPwmMatches()`, `countPwmMatches()`

Examples

```
## Load the example peaks & the sequences
data("ar_er_peaks")
data("ar_er_seq")
sq <- seqinfo(ar_er_peaks)
## Now sample size-matched ranges 10 times larger. In real-world analyses,
## this set should be sampled as at least 1000x larger, ensuring features
## are matched to your requirements. This example masks regions with known N
## content, including centromeres & telomeres
data("hg19_mask")
set.seed(305)
bg_ranges <- makeRMRanges(
  ar_er_peaks, GRanges(sq)[1], exclude = hg19_mask, n_iter = 10
)

## Convert ranges to DNAStringSets
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19
bg_seq <- getSeq(genome, bg_ranges)

## Test for enrichment of the ESR1 motif
data("ex_pfm")
esr1 <- ex_pfm$ESR1
testMotifEnrich(esr1, ar_er_seq, bg_seq, model = "poisson")

## Test all motifs
```

```
testMotifEnrich(ex_pfm, ar_er_seq, bg_seq, model = "poisson")
```

`testMotifPos` *Test for a Uniform Distribution across a set of best matches*

Description

Test for a Uniform Distribution across a set of best matches

Usage

```
testMotifPos(
  x,
  stringset,
  binwidth = 10,
  abs = FALSE,
  rc = TRUE,
  min_score = "80%",
  break_ties = "all",
  alt = c("greater", "less", "two.sided"),
  sort_by = c("p", "none"),
  mc.cores = 1,
  ...
)
```

Arguments

`x`	A Position Weight Matrix, universalmotif object or list thereof. Alternatively can be a single DataFrame or list of DataFrames as returned by getPwmMatches with `best_only = TRUE`
`stringset`	An XStringSet. Not required if matches are supplied as x
`binwidth`	Width of bins across the range to group data into
`abs`	Use absolute positions around zero to find symmetrical enrichment
`rc`	logical(1) Also find matches using the reverse complement of each PWM
`min_score`	The minimum score to return a match
`break_ties`	Choose how to resolve matches with tied scores
`alt`	Alternative hypothesis for the binomial test
`sort_by`	Column to sort results by
`mc.cores`	Passed to mclapply
`...`	Passed to matchPWM

Details

This function tests for an even positional spread of motif matches across a set of sequences, using the assumption (i.e. H~0~) that if there is no positional bias, matches will be evenly distributed across all positions within a set of sequences. Conversely, if there is positional bias, typically but not necessarily near the centre of a range, this function intends to detect this signal, as a rejection of the null hypothesis.

Input can be provided as the output from getPwmMatches setting `best_only = TRUE` if these matches have already been identified. If choosing to provide this object to the argument `matches`, nothing is required for the arguments `pwm`, `stringset`, `rc`, `min_score` or `break_ties` Otherwise, a Position Weight Matrix (PWM) and an `XStringSet` are required, along with the relevant arguments, with best matches identified within the function.

The set of best matches are then grouped into bins along the range, with the central bin containing zero, and tallied. Setting `abs` to `TRUE` will set all positions from the centre as *absolute values*, returning counts purely as bins with distances from zero, marking this as an inclusive lower bound. Motif alignments are assigned into bins based on the central position of the match, as provided in the column `from_centre` when calling getPwmMatches.

The binom.test is performed on each bin using the alternative hypothesis, with the returned p-values across all bins combined using the Harmonic Mean p-value (HMP) (See p.hmp). All bins with raw p-values below the HMP are identified and the returned values for start, end, centre, width, matches in region, expected and enrichment are across this set of bins. The expectation is that where a positional bias is evident, this will be a narrow range containing a non-trivial proportion of the total matches.

It should also be noted that `binom.test()` can return p-values of zero, as beyond machine precision. In these instances, zero p-values are excluded from calculation of the HMP. This will give a very slight conservative bias, and assumes that for these extreme cases, neighbouring bins are highly likely to also return extremely low p-values and no significance will be lost.

Value

A data.frame with columns `start`, `end`, `centre`, `width`, `total_matches`, `matches_in_region`, `expected`, `enrichment`, `prop_total`, `p` and `consensus_motif` The total matches represent the total number of matches within the set of sequences, whilst the number observed in the final region are also given, along with the proportion of the total this represents. Enrichment is simply the ratio of observed to expected based on the expectation of the null hypothesis

The consensus motif across all matches is returned as a Position Frequency Matrix (PFM) using consensusMatrix.

Examples

```
## Load the example PWM
data("ex_pfm")
esr1 <- ex_pfm$ESR1

## Load the example sequences
data("ar_er_seq")

## Get the best match and use this data
matches <- getPwmMatches(esr1, ar_er_seq, best_only = TRUE)
## Test for enrichment in any position
testMotifPos(matches)

## Provide a list of PWMs, testing for distance from zero
```

```
testMotifPos(ex_pfm, ar_er_seq, abs = TRUE, binwidth = 10)
```

`zr75_enh` *Candidate Enhancer Regions from ZR-75-1 Cells*

Description

The chr1 subset of candidate enhancers for ZR-75-1 cells

Usage

```
data("zr75_enh")
```

Format

An object of class `GRanges` of length 5237.

Details

These enhancers are the chr1 subset of enhancer regions for ZR-75-1 cells as identified by EnhancerAtlas 2.0

#' Generation of these ranges is documented in `system.file("scripts/zr75_enh.R", package = "motifTestR")`

Source

http://www.enhanceratlas.org/index.php

Examples

```
data("zr75_enh")
zr75_enh
```

Index