Package 'metaMix'

February 11, 2019

Title Bayesian Mixture Analysis for Metagenomic Community Profiling

Version 0.3

Author Sofia Morfopoulou <sofia.morfopoulou.10@ucl.ac.uk>

Maintainer Sofia Morfopoulou <sofia.morfopoulou.10@ucl.ac.uk>

Depends R (>= 3.2)

Imports data.table (>= 1.9.2), Matrix, gtools, Rmpi, ggplot2

Suggests knitr

VignetteBuilder knitr

Description Resolves complex metagenomic mixtures by analysing
deep sequencing data, using a mixture model based approach.
The use of parallel Monte Carlo Markov chains for the exploration
of the species space enables the identification of the set
of species more likely to contribute to the mixture.

License GPL-3

LazyData true

SystemRequirements Open MPI (>=1.4.3)

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-02-11 16:20:03 UTC

R topics documented:

`bayes.model.aver` *Bayesian Model Averaging*

Description

Perform Bayesian Model Averaging. We concentrate on the chain with temperature=1 , i.e the untempered posterior, to study the distribution over the model choices and perform model averaging. We consider as present the species that have a posterior probability greater than 0.9. We then fit the mixture model with these species in order to obtain relative abundances and read classification probabilities. A tab seperated file that has a species summary is produced, as well as log-likelihood traceplots and cumulative histogram plots.

bayes.model.aver.explicit is the same function as bayes.model.aver with a more involved syntax.

Usage

```
bayes.model.aver(step2, step3, taxon.name.map = NULL,
  poster.prob.thr = 0.9, burnin = 0.4)

bayes.model.aver.explicit(result, pij.sparse.mat, read.weights, outDir,
  gen.prob.unknown, taxon.name.map = NULL, poster.prob.thr = 0.9,
  burnin = 0.4)
```

Arguments

`step2`	list. The output from reduce.space(), i.e the second step of the pipeline. Alternatively, it can be a character string containing the path name of the ".RData" file where step2 list was saved.
`step3`	list. The output from parallel.temper(), i.e the third step of the pipeline. Alternatively, it can be a character string containing the path name of the ".RData" file where step3 list was saved.
`taxon.name.map`	The 'names.dmp' taxonomy names file, mapping each taxon identifier to the corresponding scientific name. It can be downloaded from `ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz`
`poster.prob.thr`	Posterior probability of presence of species threshold for reporting in the species summary.
`burnin`	Percentage of burn in iterations, default value is 0.4
`result`	The list produced by parallel.temper() (or paraller.temper.nucl()) . It holds a detailed record for each chain, what moves were proposed, which were accepted and which were rejected as well the log-likelihood through the iterations.
`pij.sparse.mat`	see ?reduce.space
`read.weights`	see ?reduce.space
`outDir`	see ?reduce.space
`gen.prob.unknown`	see ?reduce.space

Examples

```
## See vignette for more details

## Not run:
# Either load the object created by previous steps
data(step2)   ## example output of step2, i.e reduce.space()
data(step3)   ## example ouput of step3, i.e  parallel.temper()
step4<-bayes.model.aver(step2=step2, step3=step3, taxon.name.map="pathtoFile/taxon.file")

# or alternatively point to the location of the step2.RData and step3.RData objects
step4<-bayes.model.aver(step2="pathtoFile/step2.RData", step3="pathtoFile/step3.RData",
                        taxon.name.map="pathtoFile/taxon.file")


## End(Not run)
```

`generative.prob` *Compute generative probabilities from BLAST output*

Description

generative.prob() computes the probability for a read to be generated by a certain species, given that it originates from this species. The input for this function is the tabular BLAST output format, either default or custom. The function uses the recorded mismatches to produce a Poisson probability.

generative.prob.nucl() for when we have nucleotide similarity, i.e we have performed BLASTn.

Usage

```
generative.prob(blast.output.file = NULL, read.length.file = 80,
  contig.weight.file = 1, gi.taxon.file = NULL,
  protaccession.taxon.file = NULL, gi.or.prot = "prot",
  gen.prob.unknown = 1e-06, outDir = NULL, blast.default = TRUE)

generative.prob.nucl(blast.output.file = NULL, read.length.file = 80,
  contig.weight.file = 1, gi.taxon.file, gen.prob.unknown = 1e-20,
  outDir = NULL, genomeLength = NULL, blast.default = TRUE)
```

Arguments

`blast.output.file`
: This is the tabular BLASTx output format for generative.prob(), while it is the tabular BLASTn output format for generative.prob.nucl(). It can either be the default output format or a specific custom output format, incorporating read length and taxon identifier. Please see the vignette for column order and the exact BLAST command to use. You can also use DIAMOND instead of BLASTx which is much faster and produces default format according to BLAST default output specifications.

`read.length.file`
: This argument can either be a file mapping each read to its length or a numerical value, representing the average read length.

`contig.weight.file`
: This argument can either be a file where weights are assigned to reads and contigs. For unassembled reads the weight is equal to 1 while for contigs the weight should reflect the number of reads that assembled it.

`gi.taxon.file`
: For generative.prob() this would be the 'gi_taxid_prot.dmp' taxonomy file, mapping each protein gi identifier to the corresponding taxon identifier. It can be downloaded from `ftp://ftp.ncbi.nih.gov/pub/taxonomy/gi_taxid_prot.dmp.gz` . For generative.prob.nucl() this would be the 'gi_taxid_nucl.dmp' taxonomy file, mapping each nucleotide gi identifier to the corresponding taxon identifier. It can be downloaded from `ftp://ftp.ncbi.nih.gov/pub/taxonomy/gi_taxid_nucl.dmp.gz`.

`protaccession.taxon.file`
: This parameter has been added as NCBI is phasing out the usage of GI identifiers. For generative.prob() this would be the prot.accession2taxid taxonomy file, mapping each protein accession identifier to the corresponding taxon identifier. It can be downloaded from `ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/accession2taxid/prot.accession2taxid.gz`. I have found that it is useful to concatenate it with `ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/accession2taxid/dead_prot.accession2taxid.gz` so you can search in both files for the protein identifier (sometimes obsolete sequences can still be present in latest RefSeq releases but not in taxonomy files and vice versa and these mismatches can cause loss of information). TODO add support for nucleotides as well.

`gi.or.prot`
: This parameter specifies whether the user is using the GI identifiers or protein accession identifiers to map to taxon identifiers. Values are 'gi' or 'prot'. The default value is 'prot'.

`gen.prob.unknown`
: User-defined generative probability for unknown category. Default value for generative.prob() is 1e-06, while for generative.prob.nucl() is 1e-20.

`outDir`
: Output directory.

`blast.default`
: logical. Is the input the default blast output tabular format? Default value is TRUE. That means that the BLAST output file needs to have the following fields:Query id, Subject id, percent identity, alignment length, mismatches, gap openings, query start, query end, subject start, subject end, e-value, bit score. Alternatively we can use the 'blast.default=FALSE' option, providing a custom blast output that has been produced using the option -outfmt '6 qacc qlen sacc slen stitle bitscore length pident evalue staxids'.

`genomeLength`
: This is applicable only for generative.prob.nucl() . It is a file mapping each genome/nucleotide to its respective length. The file must be tab seperated and the first column the nucleotide gi identifier (integer) and the second the corresponding sequence length (integer). It will be used to correct the Poisson probabilities between each read and genome.

Value

step1: A list with five elements. The first one (pij.sparse.mat) is a sparse matrix with the generative probability between each read and each species. The second (ordered.species) is matrix containing all the potential species as recorded by BLAST, ordered by the number of reads. The third one (read.weights) is a data.frame mapping each contig to a weight which is essentially the number of reads that were used to assemble it. For unassembled reads the weight is equal to one. The fourth one is the generative probability for the unknown category (gen.prob.unknown), to be used in all subsequent steps. Finally we also record the output directory (outDir) where the results will be stored.

Examples

```
# See vignette for more details

## Not run:
# When using custom BLAST output file
step1 <-generative.prob(blast.output.file = "pathtoFile/blastOut.custom", blast.default=FALSE)

# When using default BLAST output file
step1 <-generative.prob(blast.output.file = "pathtoFile/blastOut.default",
                        read.length.file="pathtoFile/read.lengths",
                        contig.weight.file="pathtoFile/read.weights",
                        gi.taxon.file = "pathtoFile/taxon.file")

## End(Not run)
```

`parallel.temper` *Parallel Tempering MCMC*

Description

Performs Parallel Tempering MCMC to explore the species state space. Two types of moves are implemented: a mutation step (within chain) and an exchange step (between neighboring chains). If working with BLASTn data, use parallel.temper.nucl().

parallel.temper.explicit is the same function as parallel.temper but with a more involved syntax.

Usage

```
parallel.temper(step2, readSupport = 10, noChains = 12, seed = 1,
  iter = 500, bypass = FALSE)

parallel.temper.explicit(readSupport = 10, noChains = 12,
  pij.sparse.mat, read.weights, ordered.species, gen.prob.unknown, outDir,
  seed = 1, iter = 500, bypass = FALSE)
```

Arguments

`step2`	list. The output from reduce.space(). Alternatively, it can be a character string containing the path name of the ".RData" file where step2 list was saved.
`readSupport`	The number of reads the user requires in order to believe in the presence of the species. It is used to compute the penalty factor. The default value is 10. We compute the logarithmic penalty value as the log-likelihood difference between two models: one where all N reads belong to the "unknown" category and one where r reads have a perfect match to some unspecified species and the remaining reads belong to the "unknown" category.
`noChains`	The number of parallel chains to run. The default value is 12.
`seed`	Optional argument that sets the random seed (default is 1) to make results reproducible.
`iter`	The number of MCMC iterations. The default behavior of metaMix is to take into account the number of potential species after step 2 in order in order to compute the number of MCMC iterations. By default metaMix will choose the greater value between a) the user-specified value for iter and b) the product of (5 * the number of potential species). This behavior can by bypassed by setting the bypass parameter to TRUE. Then the MCMC will run for exactly the user-specified number iter.
`bypass`	A logical flag. If set to TRUE the MCMC will run for exactly "iter" iterations. If FALSE, metaMix defaults to choosing the greater value between "iter" and "5*(nrow(ordered.sepcies))".
`pij.sparse.mat`	sparse matrix of generative probabilities, see value of ?reduce.space.
`read.weights`	see ?reduce.space.
`ordered.species`	see ?reduce.space.
`gen.prob.unknown`	see ?reduce.space.
`outDir`	see ?reduce.space.

Value

step3: A list with two elements. The first one (result) is a list that records MCMC information from each parallel chain. The second one (duration) records how much time the MCMC exploration took.

See Also

`parallel.temper.nucl` This function should be used when working with BLASTn data.

Examples

```
## See vignette for more details

## Not run:
# Either load the object created by previous step (i.e from function reduce.space() )
```

```
data(step2)   ## example output of reduce.space
step3<-parallel.temper(step2=step2)

# or alternatively point to the location of the step2.RData object
step3 <- parallel.temper(step2="/pathtoFile/step2.RData")

## End(Not run)
```

`parallel.temper.nucl` *Parallel Tempering MCMC*

Description

Performs Parallel Tempering MCMC to explore the species state space. Two types of moves are implemented: a mutation step (within chain) and an exchange step (between neighboring chains). If working with BLASTx data, use parallel.temper().

parallel.temper.nucl.explicit is the same function as parallel.temper.nucl with a more involved syntax.

Usage

```
parallel.temper.nucl(step2, readSupport = 30, noChains = 12,
  seed = 1, median.genome.length = 284332)

parallel.temper.nucl.explicit(readSupport = 30, noChains = 12,
  pij.sparse.mat, read.weights, ordered.species, gen.prob.unknown, outDir,
  seed = 1, median.genome.length = 284332)
```

Arguments

`step2`	list. The output from reduce.space(). Alternatively, it can be a character string containing the path name of the ".RData" file where step2 list was saved.
`readSupport`	The number of reads the user requires in order to believe in the presence of the species. It is used to compute the penalty factor. The default value is 30. We compute the logarithmic penalty value as the log-likelihood difference between two models: one where all N reads belong to the "unknown" category and one where r reads have a perfect match to some unspecified species and the remaining reads belong to the "unknown" category.
`noChains`	The number of parallel chains to run. The default value is 12.
`seed`	Optional argument that sets the random seed (default is 1) to make results reproducible.
`median.genome.length`	To use in the penalty computation.
`pij.sparse.mat`	sparse matrix of generative probabilities, see value of ?reduce.space.
`read.weights`	see ?reduce.space.

`ordered.species`	see ?reduce.space.
`gen.prob.unknown`	see ?reduce.space.
`outDir`	see ?reduce.space.

Value

step3: A list with two elements. The first one (result) is a list that records MCMC information from each parallel chain. The second one (duration) records how much time the MCMC exploration took.

See Also

`parallel.temper` This function should be used when working with BLASTx data.

`reduce.space` *Reduce the space of potential species by fitting the mixture model with all potential species as categories*

Description

Having the generative probabilities from step1 (generative.prob() or generative.prob.nucl()), we could proceed directly with the PT MCMC to explore the state space. Typically the number of total potential species is large. Therefore we reduce the size of the state-space, by decreasing the number of species to the low hundreds. We achieve this by fitting a Mixture Model with as many categories as all the potential species. Post fitting, we retain only the species categories that are not empty, that is categories that have at least one read assigned to them.

reduce.space.explicit is the same function as reduce.space but with more involved syntax.

Usage

```
reduce.space(step1, read.cutoff = 1, EMiter = 500, seed = 1)

reduce.space.explicit(pij.sparse.mat, ordered.species, read.weights,
  outDir, gen.prob.unknown, read.cutoff = 1, EMiter = 500, seed = 1)
```

Arguments

`step1`	list. The output from generative.prob() (or generative.prob.nucl(), that is the first step of the pipeline. Alternatively, it can be a character string containing the path name of the ".RData" file where step1 list was saved.
`read.cutoff`	numeric vector. This is the used to decide which species to retain for the subsequent MCMC exploration. Default value is 1, i.e keep all species that have at least one read assigned to them. If this number is still in the low thousands as opposed to the low hundreds the user may set this to a higher number, such as 10.

`EMiter`	Number of iterations for the EM algorithm. Default value is 500.
`seed`	Optional argument that sets the random seed (default is 1) to make results reproducible.
`pij.sparse.mat`	sparse Matrix of generative probabilities computed by generative.prob() / generative.prob.nucl().
`ordered.species`	data.frame with potential species ordered by numbers of reads matching them. Computed by generative.prob().
`read.weights`	data.frame mapping each read identifier to a weight. For contigs the weight is the number of reads that were used to assemble it. For unassembled reads the weight is equal to one.
`outDir`	character vector holding the path to the output directory where the results are written.
`gen.prob.unknown`	numeric vector. This is the generative probability for the unknown category. Default value for BLASTx-analysis is 1e-06 while for BLASTn-analysis is 1e-20.

Value

step2: A list with six elements. The first one (ordered.species) is a data.frame containing all the non-empty species categories, as decided by the all inclusive mixture model, ordered by the number of reads assigned to them. The second one (pij.sparse.mat) is a sparse matrix with the generative probability between each read and each species. read.weights, gen.prob.unknown, outDir are all carried forward from the "step1" object. Finally outputEM which records the species abundances throughout the EM iterations (not used in step3 and step4).

Examples

```
## See vignette for more details.

## Not run:
# Either load the object created by previous step
data(step1)  ## example output of step1, i.e generative.prob()
step2 <- reduce.space(step1=step1)

# or alternatively point to the location of the step1.RData object
step2 <- reduce.space(step1="/pathtoFile/step1.RData")

## End(Not run)
```

`step1` *Example output of generative.prob() for use in the vignette/examples*

Description

Example output of generative.prob() for use in the vignette/examples

Format

A list with 5 elements

`step2` *Example output of reduce.space() for use in the vignette/examples*

Description

Example output of reduce.space() for use in the vignette/examples

Format

A list with 6 elements

`step3` *Example output of parallel.temper() for use in the vignette/examples*

Description

Example output of parallel.temper() for use in the vignette/examples

Format

A list with 2 elements

Index