metaMix user guide

Sofia Morfopoulou

February 7, 2019

1 What is new

Version 0.2: Added Bayes Factor computation.

Version 0.3: Allow user to specify number of MCMC iterations (Step 3) and Burn-in percentage (Step 4). Due to NCBI phasing out GI numbers, added support for protein accession identifier file for mapping to taxon (STEP1). Fixed bug in step4 for Bayes Factor calculation when only one species is present, that is only human and unknown. Also fixed bug with semi-colon separated taxonids in custom BLAST format, as well as NAs (STEP 1). Replaced deprecated cBind function with cbind.

2 Installation

You will need to have openMPI (Message Passage Interface) installed to be able to install the R package `Rmpi`, which provides the interface to openMPI. `Rmpi` is one of the package dependencies, along with `data.table`, `Matrix`, `gtools` and `ggplot2`. You can check whether you have openMPI installed using the command `mpirun` and you can find more information here:
`http://www.open-mpi.org/software/ompi`

3 Introduction

metaMix is a tool designed to identify the set of species most likely to be present in a metagenomic community. metaMix also estimates their relative abundances and resolves ambiguous assignments by considering all reads simultaneously.

metaMix considers the competing models that could accommodate our observed data, i.e the BLASTx results and compares them. The different mixture models represent different sets of species being present in the sample. The method is structured in the following manner: in the first instance we assume that a set of species is present in the sample and we estimate the parameters given the data. At the next step, we randomly add or remove a species and fit this new model. The process is iterated in order to explore the model state space and we record the MCMC choices over time. Additionally we parallelise the process, running n (usually 12) parallel chains, allowing exchange of information between them. Using this Bayesian Mixture Model framework, we finally perform model averaging in order to account for model uncertainty.

The initial motivation for developing metaMix was the analysis of deep transcriptome sequencing datasets, with a particular focus on viral pathogen detection. However the ideas are applicable more generally to all types of metagenomics mixtures.

Some bionformatics processing is required prior to using metaMix. This is usually filtering out the low quality, duplicate and host reads. The user may wish to attempt some assembly step as well prior to resolving the mixture; however this step is not necessary. More importantly, the similarities between the short reads (and/or contigs) and a reference database must be provided. At the end of the analysis, the user will obtain a probabilistic summary of present species and some supporting plots.

The implementation of the ideas described here is computationally intensive and requires a supercomputer. However for the purposes of this tutorial, we demonstrate the usage on a toy example and all the steps can be performed on a single machine.

4 Tutorial

Step1

The work described here is similarity-based, therefore the starting point is to obtain the sequence similarity between a query and a target sequence. The obvious choice for this is BLAST. Both nucleotide and amino acid similarities are supported. We demonstrate the use of metaMix working with the latter, i.e we have used BLASTx.

Default BLAST output The default output tabular file is supported, obtained using `-outfmt 6` in the BLAST command.

```
 blastx -db referenceDB -query input.fa  -outfmt 6  -max_target_seqs 10
```

The default output file has the following fields:
`Query ID`, `Subject ID`, `% Identity`, `Alignment Length`, `Mismatches`, `Gap Openings`, `Query Start`, `Query End`, `Subject Start`, `Subject End`, `E-value`, `Bit Score`.

```
library(metaMix)
###Location of input files.
datapath <- system.file("extdata", package="metaMix")
blastOut.default<-file.path(datapath, "blastOut_default.tab")
read.table(blastOut.default, nrows=2, sep="\t")

##                                 V1                               V2    V3 V4 V5
## 1 NODE_547_length_108_cov_1.888889 gi|490323013|ref|WP_004212498.1| 92.16 51  4
## 2 NODE_547_length_108_cov_1.888889 gi|493738276|ref|WP_006687451.1| 57.14 42 18
##   V6 V7  V8  V9 V10     V11  V12
## 1  0  2 154 423 473 7.0e-13 67.0
## 2  0 23 148 413 454 8.4e-02 34.7
```

metaMix needs information on the read lengths as well as a file mapping the gi identifiers to the taxon identifiers. These are not included in the default output of BLAST, therefore should be provided as additional arguments.

```
read.lengths<-file.path(datapath, "read_lengths.tab")
read.weights<-file.path(datapath, "read_weights.tab")
taxon.file<-file.path(datapath, "gi_taxid_prot_example.dmp")

read.table(read.lengths, nrows=2, sep="\t")

##                                 V1  V2
## 1 NODE_427_length_161_cov_1.267081 209
## 2 NODE_428_length_114_cov_1.745614 162

read.table(read.weights, nrows=2, sep="\t")

##                                 V1 V2
## 1 NODE_476_length_207_cov_3.835749 10
## 2 NODE_524_length_423_cov_5.368794 26

read.table(taxon.file, nrows=2, sep="\t")

##        V1    V2
## 1 9625360 10849
## 2 9625363 10849
```

Custom BLAST output Alternatively, metaMix accepts a custom BLAST output file that has already incorporated the read lengths and the taxon identifiers. At the moment, only the output that is produced by the following command is supported:

```
  blastx -db referenceDB -query input.fa -max_target_seqs 10
-outfmt "6 qacc qlen sacc slen mismatch bitscore length pident evalue staxids"
```

Therefore the fields are
`Query ID`, `Query Length`, `Subject ID`, `Subject Length`, `Mismatches`, `Bit Score`, `Alignment Length`, `%Identity`, `E-value`, `Taxon ID`.

```
blastOut.custom<-file.path(datapath, "blastOut_custom.tab")
read.table(blastOut.custom, nrows=2, sep="\t")

##                                    V1  V2                              V3  V4
## 1 NODE_61_length_112_cov_1177.339233 160 gi|446388692|ref|WP_000466547.1| 328
## 2 NODE_61_length_112_cov_1177.339233 160      gi|9626383|ref|NP_040713.1| 328
##   V5   V6 V7  V8    V9    V10
## 1  0 70.1 52 100 2e-14      2
## 2  0 70.1 52 100 2e-14 374840
```

The first step in the analysis is to compute the read-species generative probabilities based on the BLASTx data. We achieve this by using the `generative.prob()` function. In this instance we will work with the custom BLAST output file.

```
  step1 <-generative.prob(blast.output.file = blastOut.custom,
                          contig.weight.file=read.weights,
                          blast.default=FALSE,
                          outDir=NULL)

## Warning in rm(blast.length):  object 'blast.length' not found
```

where `blast.default` denotes whether we are working with the BLAST default output (TRUE) or with the specified above custom output (FALSE).

`blast.output.file` is the tabular BLASTx output file.

If we are working with unassembled reads, we can omit the argument `contig.weight.file` as the weight is set by default to be 1, same for all reads. However if an assembly step has been performed, as in this example, we need to provide information on the number of reads that make up each contig. This will be a two column tab-separated file, where the first column is the contig identifier and the second the number of reads.

Finally `outDir` is the directory where the results are written and where an object from each step is saved. When it is set to NULL no objects will be saved.

NOTE: If we were using the default BLAST output the command would look like so:

```
step1 <-generative.prob(blast.output.file = blastOut.default,
                        read.length.file=read.lengths,
                        contig.weight.file=read.weights,
                        gi.taxon.file = taxon.file,
                        blast.default=TRUE,
                        outDir=NULL)
```

The information missing from the BLAST file is now provided with two extra arguments: `read.length.file` can be either the file mapping each read to its sequence length or a numerical value, representing the average read length (default value=100).

`'gi_taxid_prot.dmp'` is a 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

The function `generative.prob` creates a list of five elements. One of these is a sparse matrix `pij.sparse.mat` where each row corresponds to one read and each column to a species. The value of the cell is the generative probability p_{ij}. Additionally a `data.frame` with all the species that correspond to the proteins in the BLASTx output file. Finally the `read.weights`, `gen.prob.unknown` and `outDir` are the other three elements of the list `step1`, carried forward to be used in the second step.

```
###The resulting list consists of five elements
names(step1)

## [1] "ordered.species" "pij.sparse.mat"  "read.weights"    "outDir"
## [5] "gen.prob.unknown"

### The sparse matrix of generative probs.
step1$pij.sparse.mat[1:5,c("374840",  "258", "unknown")]

## 5 x 3 sparse Matrix of class "dgCMatrix"
##                                                              374840 258 unknown
## @M01520:37:000000000-A4TDP:1:1101:13751:20874_1:N:0:1 .                .   1e-06
## @M01520:37:000000000-A4TDP:1:1101:13751:20874_2:N:0:1 .                .   1e-06
## @M01520:37:000000000-A4TDP:1:1101:13805:1480_1:N:0:1  7.366052e-05    .   1e-06
## @M01520:37:000000000-A4TDP:1:1101:15097:1496_1:N:0:1  1.000000e+00    .   1e-06
## @M01520:37:000000000-A4TDP:1:1101:16186:1569_1:N:0:1  9.389005e-01    .   1e-06

### There are that many potential species in the sample:
nrow(step1$ordered.species)

## [1] 224
```

Step2

Having the generative probabilities from the previous step (generative.prob), we could proceed directly with the PT MCMC to explore the state space. However, typically the number of all potential species S is large. We are therefore interested in reducing the size of the species pool, from the thousands to the low hundreds.

In this simple example we have only 224 organisms but still we attempt to reduce it for demonstrating the usage of the function. We achieve this by fitting a mixture model with 224 categories, considering all 224 potential species simultaneously. Post fitting, we retain only the species categories that are not empty, that is the categories that have at least one read assigned to them. The required argument is simply the list created in the first step, i.e using the `generative.prob` function.

```
step2 <- reduce.space(step1=step1)
```

Alternatively, if the list created in the first step was saved in a "step1.RData" file, a character string containing the path to the file could be provided, i.e

```
step2 <- reduce.space(step1="/pathtoFile/step1.RData")
```

To speed up computations, we have already performed step2 and saved the output which we will now load:

```
##These are the elements of the step2 list.
names(step2)

## [1] "outputEM"         "pij.sparse.mat"   "ordered.species"  "read.weights"
## [5] "outDir"           "gen.prob.unknown"

## After this approximating step, there are now that many potential species in
##the sample:
nrow(step2$ordered.species)

## [1] 7

## And these are:
step2$ordered.species

##        taxonID countReads samplingWeight
## 374840  374840       1888    0.165016747
## 2            2        531    0.165016747
## 28090    28090         93    0.034611090
## 13690    13690         62    0.023074060
## 645687  645687         46    0.017119464
## 258        258          2    0.004019352
## 1035      1035          2    0.004019352
```

We see that even though we started with 224 potential organisms, we reduced the species space to 7. Bear in mind that this a simple example and the usual scenario is to move from thousands of species to hundreds.

Step3

In this step, the different models are considered and compared. The space exploration by the parallel tempering MCMC is implemented by the function `parallel.temper`:

```
step3<-parallel.temper(step2=step2)
```

The required argument is simply the list created in the second step (or the character string containing the path to the respective .RData file where the step2 list was saved to), i.e using the `reduce.space` function.

An important optional argument of this function is `readSupport`. For the type of data we analyse (i.e from mostly sterile human tissues) we expect that parsimonious models with a limited number of species are more likely. Therefore our default model prior uses a penalty limiting the number of species in the model. We approximate this penalty factor based on `readSupport`, which represents the species read support required from the user in order to believe in the presence of a species in the sample. The default value is 10 and it is suitable for when we want to detect rare signal. We have found this value to work well in most human RNA-seq datasets.

Same as before, we have already performed step3 and saved the output which we now load:

```
##These are the elements of the step3 list.
names(step3)

## [1] "result"   "duration"
```

```
## Steps MCMC took during some iterations.
step3$result$slave1$record[10:15,]

##    Iter   Move Candidate Species 374840 2 28090 13690 645687 258 1035 unknown
## 10   10 Remove              645687       1 0      1      1       1   0    0        1
## 11   11 Remove              374840       1 0      1      1       1   0    0        1
## 12   12    Add                   2       1 1      1      1       1   0    0        1
## 13   13    Add                 258       1 1      1      1       1   0    0        1
## 14   14 Remove               13690       1 0      1      0       1   0    0        1
## 15   15    Add                   2       1 0      1      0       1   0    0        1
##          logL
## 10 -2988.214
## 11 -2988.214
## 12 -3070.453
## 13 -3070.453
## 14 -3504.626
## 15 -3504.626
```

For each parallel chain, the MCMC trajectory has been recorded. There is information on what steps were proposed, which were accepted or rejected throughout the iterations. For example at iteration 10, removing species 645687 was proposed but not accepted, as denoted by the 1 in the column 645687.

We can also see that between iterations 13 and 14 an exchange of the sets of species between Chain 1 and Chain 2 occurred. At iter. 13 species 2 was present, while at the next one, it no longer is there. That means that the attempt at swapping the values of the two neighboring chains was successful. This information is also recorded, i.e how many swaps were attempted and how many accepted.

Each chain will produce a log file that will be printed in your working directory.

Step4

Having explored the different possible models, the final step is to perform model averaging. We study the MCMC choices for Chain 1 and produce a probabilistic summary for the presence of the species.

```
## Location of the taxonomy names file.
taxon.file<-file.path(datapath, "names_example.dmp")

step4<-bayes.model.aver(step2=step2,
                        step3=step3,
                        taxon.name.map=taxon.file)
```

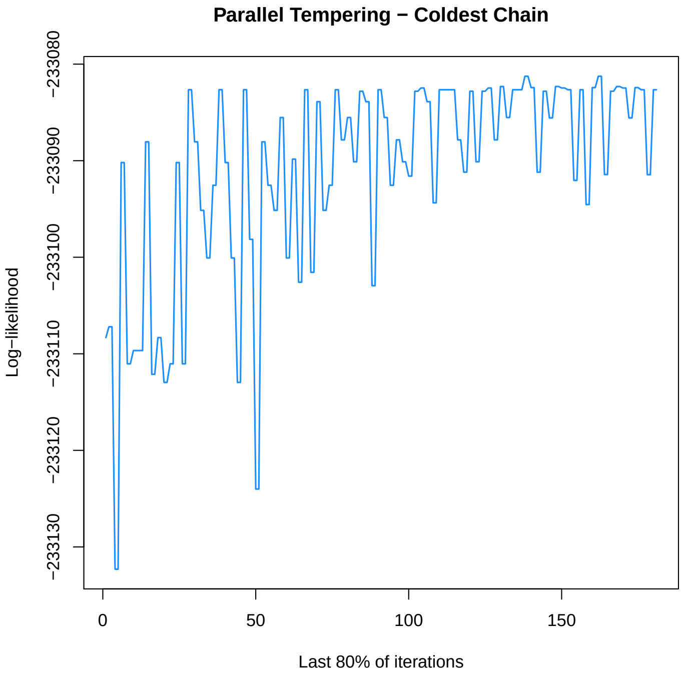
The required arguments are the lists created in the second and third steps, i.e using the `reduce.space` and the `parallel.temper` functions. Additionally the taxonomy names file 'names.dmp', which can be downloaded and extracted from `ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz`

```
##These are the elements of the step4 list.
names(step4)

## [1] "result"                  "pij.sparse.mat"          "presentSpecies.allInfo"
## [4] "output100"               "assignedReads"           "classProb"

##This is the species summary
print(step4$presentSpecies.allInfo)

##   taxonID                              scientName finalAssignments poster.prob
## 3  374840 Enterobacteria phage phiX174 sensu lato             2419        1.00
## 2   28090                     Acinetobacter lwoffii               93        1.00
## 5 unknown                                 unknown               68        1.00
## 1   13690                    Sphingobium yanoikuyae               60        0.91
## 4  645687                            Astrovirus VA1               46        1.00
##      log10BF
## 3 14316.4036
## 2   442.5012
## 5         NA
## 1   224.1779
## 4   142.6596
```

We find four species with a posterior probability greater than 0.9 (default value), plus the unknown category.

Finally we also produce log-likelihood traceplots for Chain 1. We discard the first 20% of the iterations as burn-in and we look at the mixing of the chain.

Due to having very few iterations for this toy example, the produced traceplot would not be representative or insightful. Instead we present below the log-likelihood traceplot from a real dataset.



5 Submit jobs on cluster compute servers

In order to run steps 1, 2 and 4 of metaMix (i.e `generative.prob`, `reduce.space`, `bayes.model.aver`) efficiently, these should be submitted as jobs to a compute cluster. In our experience, 4G of memory, 1 hour of wall clock time and 1 processor should be plenty. In order to run the parallel tempering efficiently (step3), we need at least 12 parallel chains. I usually request 6 or 12 threads, each with 2G of RAM. The wall clock time depends on how many iterations will be performed. A larger number of reads mean that the computations will become slower.

We typically submit all 4 steps in one go using one submission script. I usually assess the size of the dataset, but 12 hours for all 4 steps should be safe.

This is a sample submission script, it requests 6 processors on 1 node for 12 hours (more processors, less time necessary).

```
#!/bin/bash
#$ -S /bin/bash
#$ -o cluster/out
#$ -e cluster/error
#$ -cwd
#$ -pe smp 6
#$ -l tmem=2G,h_vmem=2G
#$ -l h_rt=12:00:00
#$ -V
#$ -R y

mpirun -np 1 R-3.0.1/bin/R --slave CMD BATCH --no-save --no-restore allsteps.R
```

an example allsteps.R looks like this:

```
library(metaMix)

step1<-generative.prob(blast.output.file="sample_diamond.tab",  read.length.file="read_lengths.tab",

step2<-reduce.space(step1="step1.RData")

step3<-parallel.temper(step2="step2.RData")

step4<-bayes.model.aver(step2="step2.RData", step3="step3.RData", taxon.name.map="names.dmp")
```

6 Technical information about the R session

```
sessionInfo()

## R version 3.5.1 (2018-07-02)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.10
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] metaMix_0.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.0        knitr_1.21        magrittr_1.5      munsell_0.5.0
##  [5] colorspace_1.4-0  lattice_0.20-35   rlang_0.3.1       stringr_1.3.1
##  [9] highr_0.7         plyr_1.8.4        tools_3.5.1       parallel_3.5.1
## [13] grid_3.5.1        data.table_1.12.0 gtable_0.2.0      xfun_0.4
## [17] gtools_3.8.1      digest_0.6.18     lazyeval_0.2.1    tibble_2.0.1
## [21] crayon_1.3.4      Matrix_1.2-14     ggplot2_3.1.0     evaluate_0.12
## [25] labeling_0.3      stringi_1.2.4     Rmpi_0.6-9        compiler_3.5.1
## [29] pillar_1.3.1      scales_1.0.0      pkgconfig_2.0.2
```