`metagenomeSeq`: Statistical analysis for sparse high-throughput sequencing

Joseph Nathaniel Paulson

Applied Mathematics & Statistics, and Scientific Computation
Center for Bioinformatics and Computational Biology
University of Maryland, College Park

jpaulson@umiacs.umd.edu

Modified: May 18, 2015. Compiled: April 17, 2016

Contents

1 Introduction

This is a vignette for pieces of an association study pipeline. For a full list of functions available in the package: help(package=metagenomeSeq). For more information about a particular function call: ?function. See *fitFeatureModel* for our latest development.

To load the metagenomeSeq library:

```
library(metagenomeSeq)
```

Metagenomics is the study of genetic material targeted directly from an environmental community. Originally focused on exploratory and validation projects, these studies now focus on understanding the differences in microbial communities caused by phenotypic differences. Analyzing high-throughput sequencing data has been a challenge to researchers due to the unique biological and technological biases that are present in marker-gene survey data.

We present a R package, `metagenomeSeq`, that implements methods developed to account for previously unaddressed biases specific to high-throughput sequencing microbial marker-gene survey data. Our method implements a novel normalization technique and method to account for sparsity due to undersampling. Other methods include White *et al.*'s Metastats and Segata *et al.*'s LEfSe. The first is a non-parametric permutation test on *t*-statistics and the second is a non-parametric Kruskal-Wallis test followed by subsequent wilcox rank-sum tests on subgroups to guard against positive discoveries of differential abundance driven by potential confounders - neither address normalization nor sparsity.

This vignette describes the basic protocol when using `metagenomeSeq`. A normalization method able to control for biases in measurements across taxanomic features and a mixture model that implements a zero-inflated Gaussian distribution to account for varying depths of coverage are implemented. Using a linear model methodology, it is easy to include confounding sources of variability and interpret results. Additionally, visualization functions are provided to examine discoveries.

The software was designed to determine features (be it Operational Taxanomic Unit (OTU), species, etc.) that are differentially abundant between two or more groups of multiple samples. The software was also designed to address the effects of both normalization and undersampling of microbial communities on disease association detection and testing of feature correlations.

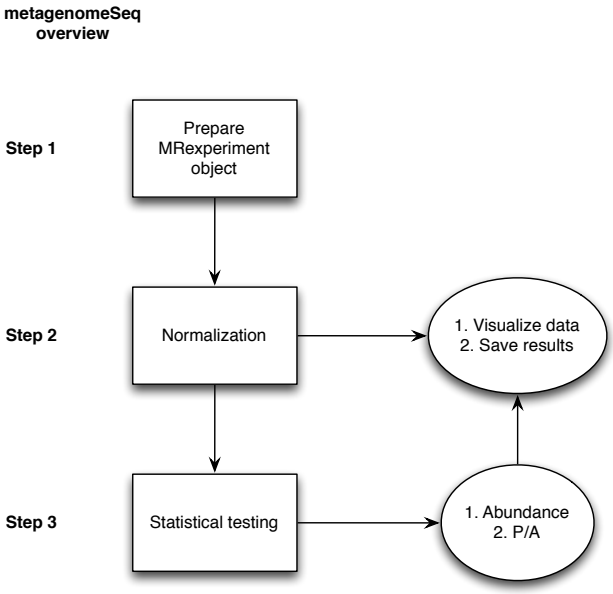

Figure 1: General overview. metagenomeSeq requires the user to convert their data into MRexperiment objects. Using those MRexperiment objects, one can normalize their data, run statistical tests (abundance or presence-absence), and visualize or save results.

2 Data preparation

Microbial marker-gene sequence data is preprocessed and counts are algorithmically defined from project-specific sequence data by clustering reads according to read similarity. Given m features and n samples, the elements in a count matrix $\mathbf{C}$ (m, n), c_{ij}, are the number of reads annotated for a particular feature i (whether it be OTU, species, genus, etc.) in sample j.

$$\begin{array}{c} \\ feature_1 \\ feature_2 \\ \vdots \\ feature_m \end{array} \begin{array}{c} \begin{array}{cccc} sample_1 & sample_2 & \dots & sample_n \end{array} \\ \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix} \end{array}$$

Count data should be stored in a delimited (tab by default) file with sample names along the first row and feature names along the first column.

Data is prepared and formatted as a `MRexperiment` object. For an overview of the internal structure please see Appendix A.

2.1 Biom-Format

You can load in BIOM file format data, the output of many popular tools out there, using the `load_biom` function. Tthe `biom2MRexperiment` and `MRexperiment2biom` functions serve as a gateway between the `biom-class` object defined in the **biom** package and a `MRexperiment-class` object. Currenlty, version 2.0 of the BIOM format is not supported. If you need to load a biom-2.0 file, see: https://gist.github.com/nosson/c9cdcb8cf95e7daae468.

2.2 Loading count data

Following preprocessing and annotation of sequencing data `metagenomeSeq` requires a count matrix with features along rows and samples along the columns. `metagenomeSeq` includes functions for loading delimited files of counts `load_meta` and phenodata `load_phenoData`.

As an example, a portion of the lung microbiome [**?**] OTU matrix is provided in `metagenomeSeq`'s library "extdata" folder. The OTU matrix is stored as a tab delimited file. `load_meta` loads the taxa and counts into a list.

```
dataDirectory <- system.file("extdata", package = "metagenomeSeq")
lung = load_meta(file.path(dataDirectory, "CHK_NAME.otus.count.csv"))
dim(lung$counts)

## [1] 1000   78
```

2.3 Loading taxonomy

Next we want to load the annotated taxonomy. Check to make sure that your taxa annotations and OTUs are in the same order as your matrix rows.

```
taxa = read.delim(file.path(dataDirectory, "CHK_otus.taxonomy.csv"),
    stringsAsFactors = FALSE)
```

As our OTUs appear to be in order with the count matrix we loaded earlier, the next step is to load phenodata.

Warning: features need to have the same names as the rows of the count matrix when we create the MRexperiment object for provenance purposes.

2.4 Loading metadata

Phenotype data can be optionally loaded into `R` with `load_phenoData`. This function loads the data as a list.

```
clin = load_phenoData(file.path(dataDirectory, "CHK_clinical.csv"),
    tran = TRUE)
ord = match(colnames(lung$counts), rownames(clin))
clin = clin[ord, ]
head(clin[1:2, ])

##                                          SampleType
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronch2.PreWash
## CHK_6467_E3B11_OW_V1V2                           OW
##                                        SiteSampled
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronchoscope.Channel
## CHK_6467_E3B11_OW_V1V2                    OralCavity
##                                     SmokingStatus
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2        Smoker
## CHK_6467_E3B11_OW_V1V2                     Smoker
```

Warning: phenotypes must have the same names as the columns on the count matrix when we create the MRexperiment object for provenance purposes.

2.5 Creating a `MRexperiment` object

Function `newMRexperiment` takes a count matrix, phenoData (annotated data frame), and featureData (annotated data frame) as input. `Biobase` provides functions to create annotated data frames. Library sizes (depths of coverage) and normalization factors are also optional inputs.

```
phenotypeData = AnnotatedDataFrame(clin)
phenotypeData

## An object of class 'AnnotatedDataFrame'
##   rowNames: CHK_6467_E3B11_BRONCH2_PREWASH_V1V2
##     CHK_6467_E3B11_OW_V1V2 ... CHK_6467_E3B09_BAL_A_V1V2
##     (78 total)
##   varLabels: SampleType SiteSampled SmokingStatus
##   varMetadata: labelDescription
```

A feature annotated data frame. In this example it is simply the OTU numbers, but it can as easily be the annotated taxonomy at multiple levels.

```
OTUdata = AnnotatedDataFrame(taxa)
OTUdata

## An object of class 'AnnotatedDataFrame'
##   rowNames: 1 2 ... 1000 (1000 total)
##   varLabels: OTU Taxonomy ... strain (10 total)
##   varMetadata: labelDescription
```

```
obj = newMRexperiment(lung$counts,phenoData=phenotypeData,featureData=OTUdata)
# Links to a paper providing further details can be included optionally.
# experimentData(obj) = annotate::pmid2MIAME("21680950")
obj

## MRexperiment (storageMode: environment)
## assayData: 1000 features, 78 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: CHK_6467_E3B11_BRONCH2_PREWASH_V1V2
##     CHK_6467_E3B11_OW_V1V2 ... CHK_6467_E3B09_BAL_A_V1V2
##     (78 total)
##   varLabels: SampleType SiteSampled SmokingStatus
##   varMetadata: labelDescription
## featureData
##   featureNames: 1 2 ... 1000 (1000 total)
##   fvarLabels: OTU Taxonomy ... strain (10 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

2.6 Example datasets

There are two datasets included as examples in the `metagenomeSeq` package. Data needs to be in a `MRexperiment` object format to normalize, run statistical tests, and visualize. As an example, throughout the vignette we'll use the following datasets. To understand a function's usage or included data simply enter ?functionName.

1. Human lung microbiome [**?**]: The lung microbiome consists of respiratory flora sampled from six healthy individuals. Three healthy nonsmokers and three healthy smokers. The upper lung tracts were sampled by oral wash and oro-/nasopharyngeal swabs. Samples were taken using two bronchoscopes, serial bronchoalveolar lavage and lower airway protected brushes.

```
data(lungData)
lungData

## MRexperiment (storageMode: environment)
## assayData: 51891 features, 78 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: CHK_6467_E3B11_BRONCH2_PREWASH_V1V2
##     CHK_6467_E3B11_OW_V1V2 ... CHK_6467_E3B09_BAL_A_V1V2
##     (78 total)
##   varLabels: SampleType SiteSampled SmokingStatus
##   varMetadata: labelDescription
## featureData
##   featureNames: 1 2 ... 51891 (51891 total)
##   fvarLabels: taxa
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
##   pubMedIds: 21680950
## Annotation:
```

2. Humanized gnotobiotic mouse gut [**?**]: Twelve germ-free adult male C57BL/6J mice were fed a low-fat, plant polysaccharide-rich diet. Each mouse was gavaged with healthy adult human fecal material. Following the fecal transplant, mice remained on the low-fat, plant polysacchaaride-rich diet for four weeks, following which a subset of 6 were switched to a high-fat and high-sugar diet for eight weeks. Fecal samples for each mouse went through PCR amplification of the bacterial 16S rRNA gene V2 region weekly. Details of experimental protocols and further details of the data can be found in Turnbaugh et. al. Sequences and further information can be found at: `http://gordonlab.wustl.edu/TurnbaughSE_10_09/STM_2009.html`

```
data(mouseData)
mouseData

## MRexperiment (storageMode: environment)
## assayData: 10172 features, 139 samples
##   element names: counts
```

```
## protocolData: none
## phenoData
##   sampleNames: PM1:20080107 PM1:20080108 ... PM9:20080303
##     (139 total)
##   varLabels: mouseID date ... status (5 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: Prevotellaceae:1 Lachnospiraceae:1 ...
##     Parabacteroides:956 (10172 total)
##   fvarLabels: superkingdom phylum ... OTU (7 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

2.7 Useful commands

Phenotype information can be accessed with the `phenoData` and `pData` methods:

```
phenoData(obj)

## An object of class 'AnnotatedDataFrame'
##   sampleNames: CHK_6467_E3B11_BRONCH2_PREWASH_V1V2
##     CHK_6467_E3B11_OW_V1V2 ... CHK_6467_E3B09_BAL_A_V1V2
##     (78 total)
##   varLabels: SampleType SiteSampled SmokingStatus
##   varMetadata: labelDescription

head(pData(obj), 3)

##                                            SampleType
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronch2.PreWash
## CHK_6467_E3B11_OW_V1V2                             OW
## CHK_6467_E3B08_OW_V1V2                             OW
##                                              SiteSampled
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronchoscope.Channel
## CHK_6467_E3B11_OW_V1V2                       OralCavity
## CHK_6467_E3B08_OW_V1V2                       OralCavity
##                                        SmokingStatus
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2        Smoker
## CHK_6467_E3B11_OW_V1V2                     Smoker
## CHK_6467_E3B08_OW_V1V2                  NonSmoker
```

Feature information can be accessed with the `featureData` and `fData` methods:

```
featureData(obj)

## An object of class 'AnnotatedDataFrame'
##   featureNames: 1 2 ... 1000 (1000 total)
##   varLabels: OTU Taxonomy ... strain (10 total)
##   varMetadata: labelDescription

head(fData(obj)[, -c(2, 10)], 3)

##   OTU superkingdom         phylum                  class
## 1   1     Bacteria Proteobacteria  Epsilonproteobacteria
## 2   2         <NA>           <NA>                   <NA>
## 3   3     Bacteria Actinobacteria Actinobacteria (class)
##               order             family         genus
## 1 Campylobacterales Campylobacteraceae Campylobacter
## 2              <NA>               <NA>          <NA>
## 3   Actinomycetales   Actinomycetaceae   Actinomyces
##                     species
## 1     Campylobacter rectus
## 2                     <NA>
## 3 Actinomyces radicidentis
```

The raw or normalized counts matrix can be accessed with the `MRcounts` function:

```
head(MRcounts(obj[, 1:2]))

##   CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 CHK_6467_E3B11_OW_V1V2
## 1                                   0                      0
## 2                                   0                      0
## 3                                   0                      0
## 4                                   0                      0
## 5                                   0                      0
## 6                                   0                      0
```

A MRexperiment-class object can be easily subsetted, for example:

```
featuresToKeep = which(rowSums(obj) >= 100)
samplesToKeep = which(pData(obj)$SmokingStatus == "Smoker")
obj_smokers = obj[featuresToKeep, samplesToKeep]
obj_smokers

## MRexperiment (storageMode: environment)
## assayData: 1 features, 33 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: CHK_6467_E3B11_BRONCH2_PREWASH_V1V2
##     CHK_6467_E3B11_OW_V1V2 ... CHK_6467_E3B09_BAL_A_V1V2
##     (33 total)
##   varLabels: SampleType SiteSampled SmokingStatus
##   varMetadata: labelDescription
## featureData
##   featureNames: 570
##   fvarLabels: OTU Taxonomy ... strain (10 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:

head(pData(obj_smokers), 3)

##                                          SampleType
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronch2.PreWash
## CHK_6467_E3B11_OW_V1V2                           OW
## CHK_6467_E3B11_BAL_A_V1V2                   BAL.A
##                                          SiteSampled
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2 Bronchoscope.Channel
## CHK_6467_E3B11_OW_V1V2                       OralCavity
## CHK_6467_E3B11_BAL_A_V1V2                          Lung
##                                     SmokingStatus
## CHK_6467_E3B11_BRONCH2_PREWASH_V1V2        Smoker
## CHK_6467_E3B11_OW_V1V2                     Smoker
## CHK_6467_E3B11_BAL_A_V1V2                  Smoker
```

Or using a threshold of minimum depth or OTU presence:

```
data(mouseData)
filterData(mouseData, present = 10, depth = 1000)

## MRexperiment (storageMode: environment)
## assayData: 1057 features, 137 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: PM1:20080108 PM1:20080114 ... PM9:20080303
##     (137 total)
##   varLabels: mouseID date ... status (5 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: Erysipelotrichaceae:8 Lachnospiraceae:129
##     ... Collinsella:34 (1057 total)
##   fvarLabels: superkingdom phylum ... OTU (7 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

3 Normalization

Normalization is required due to varying depths of coverage across samples. `cumNorm` is a normalization method that calculates scaling factors equal to the sum of counts up to a particular quantile.

Denote the lth quantile of sample j as q_j^l, that is, in sample j there are l taxonomic features with counts smaller than q_j^l. For $l = \lfloor .95m \rfloor$ then q_j^l corresponds to the 95th percentile of the count distribution for sample j.

Denote $s_j^l = \sum_{(i|c_{ij} \leq q_j^l)} c_{ij}$ as the sum of counts for sample j up to the lth quantile. Our normalization chooses a value $\hat{l} \leq m$ to define a normalization scaling factor for each sample to produce normalized counts $\tilde{c_{ij}} = \frac{c_{ij}}{s_j^{\hat{l}}} N$ where N is an appropriately chosen normalization constant. See Appendix C for more information on how our method calculates the proper percentile.

These normalization factors are stored in the experiment summary slot. Functions to determine the proper percentile `cumNormStat`, save normalized counts `exportMat`, or save various sample statistics `exportStats` are also provided. Normalized counts can be called easily by `cumNormMat(MRexperimentObject)` or `MRcounts(MRexperimentObject,norm=TRUE,log=FALSE)`

3.1 Calculating normalization factors

After defining a `MRexperiment` object, the first step is to calculate the proper percentile by which to normalize counts. There are several options in calculating and visualizing the relative differences in the reference. Figure 3 is an example from the lung dataset.

```
data(lungData)
p = cumNormStatFast(lungData)
```

To calculate the scaling factors we simply run `cumNorm`

```
lungData = cumNorm(lungData, p = p)
```

The user can alternatively choose different percentiles for the normalization scheme by specifying p.

There are other functions, including `normFactors`, `cumNormMat`, that return the normalization factors or a normalized matrix for a specified percentile. To see a full list of functions please refer to the manual and help pages.

3.2 Exporting data

To export normalized count matrices:

```
mat = MRcounts(lungData, norm = TRUE, log = TRUE)[1:5, 1:5]
exportMat(mat, file = file.path(dataDirectory, "tmp.tsv"))
```

To save sample statistics (sample scaling factor, quantile value, number of identified features and library size):

```
exportStats(lungData[, 1:5], file = file.path(dataDirectory, "tmp.tsv"))

## Default value being used.

head(read.csv(file = file.path(dataDirectory, "tmp.tsv"), sep = "\t"))
```

```
##                                Subject Scaling.factor
## 1 CHK_6467_E3B11_BRONCH2_PREWASH_V1V2             67
## 2              CHK_6467_E3B11_OW_V1V2           2475
## 3              CHK_6467_E3B08_OW_V1V2           2198
## 4           CHK_6467_E3B07_BAL_A_V1V2            836
## 5           CHK_6467_E3B11_BAL_A_V1V2           1008
##   Quantile.value Number.of.identified.features Library.size
## 1              2                             60          271
## 2              1                           3299         7863
## 3              1                           2994         8360
## 4              1                           1188         5249
## 5              1                           1098         3383
```

4 Statistical testing

Now that we have taken care of normalization we can address the effects of under sampling on the detecting differentially abundant features (OTUs, genes, etc). This is our latest development and we recommend *fitFeatureModel* over *fitZig*. *MRcoefs*, *MRtable* and *MRfulltable* are useful summary tables of the model outputs.

4.1 Zero-inflated Log-Normal mixture model for each feature

By reparametrizing our zero-inflation model, we're able to fit a zero-inflated model for each specific OTU separately. We currently recommend using the zero-inflated log-normal model as implemented in *fitFeatureModel*.

4.1.1 Example using fitFeatureModel for differential abundance testing

Here is an example comparing smoker's and non-smokers lung microbiome.

```
data(lungData)
lungData = lungData[, -which(is.na(pData(lungData)$SmokingStatus))]
lungData = filterData(lungData, present = 30, depth = 1)
lungData <- cumNorm(lungData, p = 0.5)
s <- normFactors(lungData)
pd <- pData(lungData)
mod <- model.matrix(~1 + SmokingStatus, data = pd)
lungres1 = fitFeatureModel(lungData, mod)
head(MRcoefs(lungres1))
```

```
##           logFC        se      pvalues   adjPvalues
## 3465  -4.824949 0.5697511 0.000000e+00 0.000000e+00
## 35827 -4.304266 0.5445548 2.664535e-15 1.079137e-13
## 2817   2.320656 0.4324661 8.045793e-08 1.629273e-06
## 2735   2.260203 0.4331098 1.803341e-07 2.921412e-06
## 5411   1.748296 0.3092461 1.572921e-08 4.246888e-07
## 48745 -1.645805 0.3293117 5.801451e-07 7.831959e-06
```

4.2 Zero-inflated Gaussian mixture model

The depth of coverage in a sample is directly related to how many features are detected in a sample motivating our zero-inflated Gaussian (ZIG) mixture model. Figure 2 is representative of the linear relationship between depth of coverage and OTU identification ubiquitous in marker-gene survey datasets currently available. For a quick overview of the mathematical model see Appendix B.

Function `fitZig` performs a complex mathematical optimization routine to estimate probabilities that a zero for a particular feature in a sample is a technical zero or not. The function relies heavily on the `limma` package [**?**]. Design matrices can be created in R by using the `model.matrix` function and are inputs for `fitZig`.

For large survey studies it is often pertinent to include phenotype information or confounders into a design matrix when testing the association between the abundance of taxonomic features and a phenotype phenotype of interest (disease, for instance). Our linear model methodology can easily incorporate these confounding covariates in a straightforward manner. `fitZig` output includes weighted fits for each of the m features. Results can be filtered and saved using `MRcoefs` or `MRtable`.

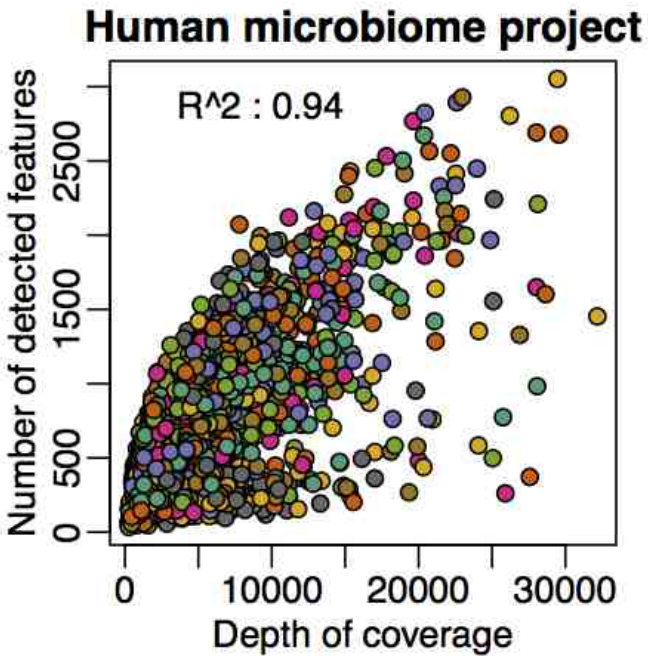

Figure 2: The number of unique features is plotted against depth of coverage for samples from the Human Microbiome Project [?]. Including the depth of coverage and the interaction of body site and sequencing site we are able to acheive an adjusted R^2 of .94. The zero-inflated Gaussian mixture was developed to account for missing features.

4.2.1 Example using fitZig for differential abundance testing

Warning: The user should restrict significant features to those with a minimum number of positive samples. What this means is that one should not claim features are significant unless the effective number of samples is above a particular percentage. For example, fold-change estimates might be unreliable if an entire group does not have a positive count for the feature in question.

We recommend the user remove features based on the number of estimated effective samples, please see `calculateEffectiveSamples`. We recommend removing features with less than the average number of effective samples in all features. In essence, setting eff = .5 when using `MRcoefs`, `MRfulltable`, or `MRtable`. To find features absent from a group the function `uniqueFeatures` provides a table of the feature ids, the number of positive features and reads for each group.

In our analysis of the lung microbiome data, we can remove features that are not present in many samples, controls, and calculate the normalization factors. The user needs to decide which metadata should be included in the linear model.

```
data(lungData)
controls = grep("Extraction.Control", pData(lungData)$SampleType)
lungTrim = lungData[, -controls]
rareFeatures = which(rowSums(MRcounts(lungTrim) > 0) < 10)
lungTrim = lungTrim[-rareFeatures, ]
lungp = cumNormStat(lungTrim, pFlag = TRUE, main = "Trimmed lung data")

## Default value being used.
```

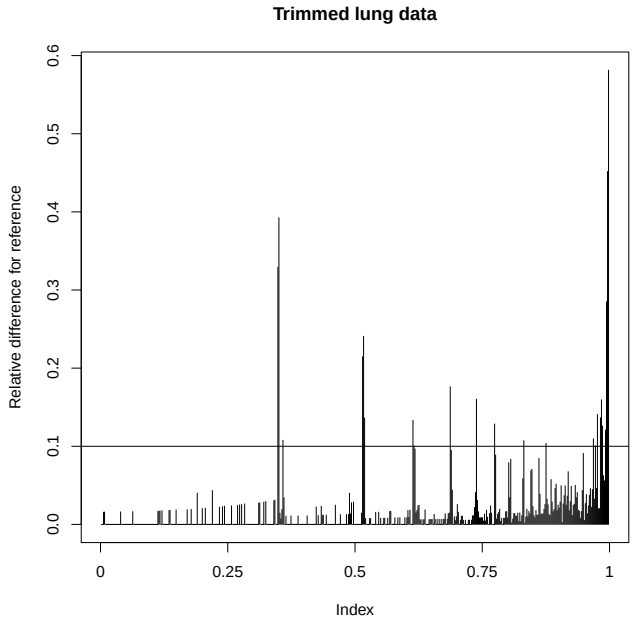

Figure 3: Relative difference for the median difference in counts from the reference.

```
lungTrim = cumNorm(lungTrim, p = lungp)
```

After the user defines an appropriate model matrix for hypothesis testing there are optional inputs to `fitZig`, including settings determined by `zigControl`. We ask the user to review the help files for both `fitZig` and `zigControl`. For this example we include body site as covariates and want to test for the bacteria differentially abundant between smokers and non-smokers.

```
smokingStatus = pData(lungTrim)$SmokingStatus
bodySite = pData(lungTrim)$SampleType
normFactor = normFactors(lungTrim)
normFactor = log2(normFactor/median(normFactor) + 1)
mod = model.matrix(~smokingStatus + bodySite + normFactor)
settings = zigControl(maxit = 10, verbose = TRUE)
fit = fitZig(obj = lungTrim, mod = mod, useCSSoffset = FALSE, control = settings

## it= 0, nll=88.42, log10(eps+1)=Inf, stillActive=1029
## it= 1, nll=93.56, log10(eps+1)=0.06, stillActive=261
## it= 2, nll=93.46, log10(eps+1)=0.05, stillActive=120
## it= 3, nll=93.80, log10(eps+1)=0.05, stillActive=22
## it= 4, nll=93.94, log10(eps+1)=0.03, stillActive=3
## it= 5, nll=93.93, log10(eps+1)=0.00, stillActive=1
## it= 6, nll=93.90, log10(eps+1)=0.00, stillActive=1
## it= 7, nll=93.87, log10(eps+1)=0.00, stillActive=1
## it= 8, nll=93.86, log10(eps+1)=0.00, stillActive=1
## it= 9, nll=93.85, log10(eps+1)=0.00, stillActive=1

# The default, useCSSoffset = TRUE, automatically includes the CSS
# scaling normalization factor.
```

The result, `fit`, is a list providing detailed estimates of the fits including a `limma` fit in `fit$fit` and an `ebayes` statistical fit in `fit$eb`. This data can be analyzed like any `limma` fit and in this example, the column of the fitted coefficientsrepresents the fold-change for our "smoker" vs. "nonsmoker" analysis.

Looking at the particular analysis just performed, there appears to be OTUs representing two *Prevotella*, two *Neisseria*, a *Porphyromonas* and a *Leptotrichia* that are differentially abundant. One should check that similarly annotated OTUs are not equally differentially abundant in controls.

Alternatively, the user can input a model with their own normalization factors including them directly in the model matrix and specifying the option `useCSSoffset = FALSE` in fitZig.

4.2.2 Multiple groups

Assuming there are multiple groups it is possible to make use of Limma's topTable functions for F-tests and contrast functions to compare multiple groups and covariates of interest. The output of fitZig includes a 'MLArrayLM' Limma object that can be called on by other functions. When running fitZig by default there is an additional covariate added to the design matrix. The fit and the ultimate design matrix are crucial for contrasts.

```
# maxit=1 is for demonstration purposes
settings = zigControl(maxit = 1, verbose = FALSE)
mod = model.matrix(~bodySite)
colnames(mod) = levels(bodySite)
# fitting the ZIG model
res = fitZig(obj = lungTrim, mod = mod, control = settings)
# The output of fitZig contains a list of various useful items.
# hint: names(res).  Probably the most useful is the limma
# 'MLArrayLM' object called fit.
zigFit = res$fit
finalMod = res$fit$design

contrast.matrix = makeContrasts(BAL.A - BAL.B, OW - PSB, levels = finalMod)
fit2 = contrasts.fit(zigFit, contrast.matrix)
fit2 = eBayes(fit2)
topTable(fit2)

##        BAL.A...BAL.B  OW...PSB   AveExpr         F      P.Value
## 18531     0.37318792  2.075648 0.7343081 12.715105 5.359780e-05
## 6291     -0.10695735  1.658829 0.4671470 12.956898 5.482439e-05
## 37977    -0.37995461  2.174071 0.4526060 12.528733 8.203239e-05
## 6901      0.17344138  1.466113 0.2435881 12.018652 1.335806e-04
## 40291     0.06892926  1.700238 0.2195735 11.803380 1.560761e-04
## 36117    -0.28665883  2.233996 0.4084024 10.571931 3.012092e-04
## 7343     -0.22859078  1.559465 0.3116465 10.090602 3.931844e-04
## 7342      0.59882970  1.902346 0.5334647  9.410984 4.901651e-04
## 1727      1.09837459 -2.160466 0.7780167  9.346013 5.027597e-04
## 40329    -0.07145998  1.481582 0.2475735  9.700136 5.259032e-04
##         adj.P.Val
## 18531 0.02813711
## 6291  0.02813711
## 37977 0.02813711
## 6901  0.03212047
```

```
## 40291 0.03212047
## 36117 0.05013569
## 7343  0.05013569
## 7342  0.05013569
## 1727  0.05013569
## 40329 0.05013569

# See help pages on decideTests, topTable, topTableF, vennDiagram,
# etc.
```

Further specific details can be found in section 9.3 and beyond of the Limma user guide. The take home message is that to make use of any Limma functions one needs to extract the final model matrix used: *resfitdesign* and the MLArrayLM Limma fit object: *res$fit*.

4.2.3 Exporting fits

Currently functions are being developed to wrap and output results more neatly, but `MRcoefs`, `MRtable`, `MRfulltable` can be used to view coefficient fits and related statistics and export the data with optional output values - see help files to learn how they differ. An important note is that the `by` variable controls which coefficients are of interest whereas `coef` determines the display.

To only consider features that are found in a large percentage of effectively positive (positive samples + the weight of zero counts included in the Gaussian mixture) use the **eff** option in the `MRtables`.

```
taxa = sapply(strsplit(as.character(fData(lungTrim)$taxa), split = ";"),
    function(i) {
        i[length(i)]
    })
head(MRcoefs(fit, taxa = taxa, coef = 2))
```

```
##                                   smokingStatusSmoker
## Neisseria polysaccharea                     -4.031612
## Neisseria meningitidis                      -3.958899
## Prevotella intermedia                       -2.927686
## Porphyromonas sp. UQD 414                   -2.675306
## Prevotella paludivivens                      2.575672
## Leptotrichia sp. oral clone FP036            2.574172
##                                        pvalues   adjPvalues
## Neisseria polysaccharea           3.927097e-11 2.959194e-08
## Neisseria meningitidis            5.751592e-11 2.959194e-08
## Prevotella intermedia             4.339587e-09 8.930871e-07
## Porphyromonas sp. UQD 414         1.788697e-07 1.357269e-05
## Prevotella paludivivens           1.360718e-07 1.272890e-05
## Leptotrichia sp. oral clone FP036 3.544957e-04 1.414122e-03
```

4.3 Time series analysis

Implemented in the `fitTimeSeries` function is a method for calculating time intervals for which bacteria are differentially abundant. Fitting is performed using Smoothing Splines ANOVA (SS-ANOVA), as implemented in the `gss` package. Given observations at multiple time points

for two groups the method calculates a function modeling the difference in abundance across all time. Using group membership permutations weestimate a null distribution of areas under the difference curve for the time intervals of interest and report significant intervals of time.

Use of the function for analyses should cite: "Finding regions of interest in high throughput genomics data using smoothing splines" Talukder H, Paulson JN, Bravo HC. (Submitted)

For a description of how to perform a time-series / genome based analysis call the `fitTimeSeries` vignette.

```
# vignette('fitTimeSeries')
```

4.4 Log Normal permutation test

Included is a standard log normal linear model with permutation based p-values permutation. We show the fit for the same model as above using 10 permutations providing p-value resolution to the tenth. The `coef` parameter refers to the coefficient of interest to test. We first generate the list of significant features.

```
coeffOfInterest = 2
res = fitLogNormal(obj = lungTrim, mod = mod, useCSSoffset = FALSE,
    B = 10, coef = coeffOfInterest)

# extract p.values and adjust for multiple testing res$p are the
# p-values calculated through permutation
adjustedPvalues = p.adjust(res$p, method = "fdr")

# extract the absolute fold-change estimates
foldChange = abs(res$fit$coef[, coeffOfInterest])

# determine features still significant and order by the
sigList = which(adjustedPvalues <= 0.05)
sigList = sigList[order(foldChange[sigList])]

# view the top taxa associated with the coefficient of interest.
head(taxa[sigList])

## [1] "Prevotella pallens"
## [2] "Anaeroglobus geminatus"
## [3] "Veillonella montpellierensis"
## [4] "Capnocytophaga sp. oral clone DZ074"
## [5] "Macrococcus caseolyticus"
## [6] "Hydrogenophilus thermoluteolus"
```

4.5 Presence-absence testing

The hypothesis for the implemented presence-absence test is that the proportion/odds of a given feature present is higher/lower among one group of individuals compared to another, and we want to test whether any difference in the proportions observed is significant. We use Fisher's exact test to create a 2x2 contingency table and calculate p-values, odd's ratios, and confidence intervals. `fitPA` calculates the presence-absence for each organism and returns a table of p-values, odd's ratios, and confidence intervals. The function will accept either a `MRexperiment`

object or matrix. `MRfulltable` when sent a result of fitZig will also include the results of `fitPA`.

```
classes = pData(mouseData)$diet
res = fitPA(mouseData[1:5, ], cl = classes)
# Warning - the p-value is calculating 1 despite a high odd's
# ratio.
head(res)
```

```
##                           oddsRatio      lower    upper   pvalues
## Prevotellaceae:1                Inf 0.01630496      Inf 1.0000000
## Lachnospiraceae:1               Inf 0.01630496      Inf 1.0000000
## Unclassified-Screened:1         Inf 0.01630496      Inf 1.0000000
## Clostridiales:1                   0 0.00000000 24.77661 0.3884892
## Clostridiales:2                 Inf 0.01630496      Inf 1.0000000
##                           adjPvalues
## Prevotellaceae:1                   1
## Lachnospiraceae:1                  1
## Unclassified-Screened:1            1
## Clostridiales:1                    1
## Clostridiales:2                    1
```

4.6 Discovery odds ratio testing

The hypothesis for the implemented discovery test is that the proportion of observed counts for a feature of all counts are comparable between groups. We use Fisher's exact test to create a 2x2 contingency table and calculate p-values, odd's ratios, and confidence intervals. `fitDO` calculates the proportion of counts for each organism and returns a table of p-values, odd's ratios, and confidence intervals. The function will accept either a `MRexperiment` object or matrix.

```
classes = pData(mouseData)$diet
res = fitDO(mouseData[1:100, ], cl = classes, norm = FALSE, log = FALSE)
head(res)
```

```
##                           oddsRatio      lower    upper   pvalues
## Prevotellaceae:1                Inf 0.01630496      Inf 1.0000000
## Lachnospiraceae:1               Inf 0.01630496      Inf 1.0000000
## Unclassified-Screened:1         Inf 0.01630496      Inf 1.0000000
## Clostridiales:1                   0 0.00000000 24.77661 0.3884892
## Clostridiales:2                 Inf 0.01630496      Inf 1.0000000
## Firmicutes:1                      0 0.00000000 24.77661 0.3884892
##                           adjPvalues
## Prevotellaceae:1           1.0000000
## Lachnospiraceae:1          1.0000000
## Unclassified-Screened:1    1.0000000
## Clostridiales:1            0.7470946
## Clostridiales:2            1.0000000
## Firmicutes:1               0.7470946
```

4.7 Feature correlations

To test the correlations of abundance features, or samples, in a pairwise fashion we have implemented `correlationTest` and `correctIndices`. The `correlationTest` function will calculate basic pearson, spearman, kendall correlation statistics for the rows of the input and report the associated p-values. If a vector of length ncol(obj) it will also calculate the correlation of each row with the associated vector.

```
cors = correlationTest(mouseData[55:60, ], norm = FALSE, log = FALSE)
head(cors)
```

```
##                                      correlation            p
## Clostridiales:11-Lachnospiraceae:35 -0.02205882 7.965979e-01
## Clostridiales:11-Coprobacillus:3    -0.01701180 8.424431e-01
## Clostridiales:11-Lactobacillales:3  -0.01264304 8.825644e-01
## Clostridiales:11-Enterococcaceae:3   0.57315130 1.663001e-13
## Clostridiales:11-Enterococcaceae:4  -0.01264304 8.825644e-01
## Lachnospiraceae:35-Coprobacillus:3   0.24572606 3.548360e-03
```

Caution: http://www.ncbi.nlm.nih.gov/pubmed/23028285

4.8 Unique OTUs or features

To find features absent from any number of classes the function `uniqueFeatures` provides a table of the feature ids, the number of positive features and reads for each group. Thresholding for the number of positive samples or reads required are options.

```
cl = pData(mouseData)[["diet"]]
uniqueFeatures(mouseData, cl, nsamples = 10, nreads = 100)
```

```
##                       featureIndices Samp. in BK Samp. in Western
## Enterococcaceae:28              2458           0                36
## Lachnospiraceae:1453            2826          16                 0
## Firmicutes:367                  4165           0                32
## Prevotellaceae:143              7030          50                 0
## Lachnospiraceae:4122            7844          15                 0
## Enterococcus:182                8384           0                34
## Lachnospiraceae:4347            8668          50                 0
## Prevotella:79                   8749          50                 0
## Prevotella:81                   8994          71                 0
## Prevotellaceae:433              9223          53                 0
##                       Reads in BK Reads in Western
## Enterococcaceae:28              0               123
## Lachnospiraceae:1453          192                 0
## Firmicutes:367                  0               112
## Prevotellaceae:143            109                 0
## Lachnospiraceae:4122          505                 0
## Enterococcus:182                0               163
## Lachnospiraceae:4347          130                 0
## Prevotella:79                 100                 0
## Prevotella:81                 370                 0
## Prevotellaceae:433            154                 0
```

5 Aggregating counts

Normalization is recommended at the OTU level. However, functions are in place to aggregate the count matrix (normalized or not), based on a particular user defined level. Using the featureData information in the MRexperiment object, calling `aggregateByTaxonomy` or `aggTax` on a MRexperiment object and declaring particular featureData column name (i.e. 'genus') will aggregate counts to the desired level with the aggfun function (default colSums). Possible aggfun alternatives include colMeans and colMedians.

```
obj = aggTax(mouseData, lvl = "phylum", out = "matrix")
head(obj[1:5, 1:5])

##                PM1:20080107 PM1:20080108 PM1:20080114
## Actinobacteria            0            3            2
## Bacteroidetes           486          921         1103
## Cyanobacteria             0            0            0
## Firmicutes              455          922         1637
## NA                        5           25            5
##                PM1:20071211 PM1:20080121
## Actinobacteria           37            0
## Bacteroidetes           607          818
## Cyanobacteria             0            0
## Firmicutes              772         1254
## NA                        8            4
```

Additionally, aggregating samples can be done using the phenoData information in the MRexperiment object. Calling `aggregateBySample` or `aggsamp` on a MRexperiment object and declaring a particular phenoData column name (i.e. 'diet') will aggregate counts with the aggfun function (default rowMeans). Possible aggfun alternatives include rowSums and rowMedians.

```
obj = aggSamp(mouseData, fct = "mouseID", out = "matrix")
head(obj[1:5, 1:5])

##                          PM1 PM10       PM11 PM12 PM2
## Prevotellaceae:1           0    0 0.00000000    0   0
## Lachnospiraceae:1          0    0 0.00000000    0   0
## Unclassified-Screened:1    0    0 0.08333333    0   0
## Clostridiales:1            0    0 0.00000000    0   0
## Clostridiales:2            0    0 0.00000000    0   0
```

The `aggregateByTaxonomy`,`aggregateBySample`, `aggTax` `aggSamp` functions are flexible enough to put in either 1) a matrix with a vector of labels or 2) a MRexperiment object with a vector of labels or featureData column name. The function can also output either a matrix or MRexperiment object.

6 Visualization of features

To help with visualization and analysis of datasets `metagenomeSeq` has several plotting functions to gain insight of the dataset's overall structure and particular individual features. An initial interactive exploration of the data can be displayed with the `display` function.

For an overall look at the dataset we provide a number of plots including heatmaps of feature counts: `plotMRheatmap`, basic feature correlation structures: `plotCorr`, PCA/MDS coordinates of samples or features: `plotOrd`, rarefaction effects: `plotRare` and contingency table style plots: `plotBubble`.

Other plotting functions look at particular features such as the abundance for a single feature: `plotOTU` and `plotFeature`, or of multiple features at once: `plotGenus`. Plotting multiple OTUs with similar annotations allows for additional control of false discoveries.

6.1 Interactive Display

Due to recent advances in the `interactiveDisplay` package, calling the `display` function on `MRexperiment` objects will bring up a browser to explore your data through several interactive visualizations. For more detailed interactive visualizations one might be interested in the shiny-phyloseq package.

```
# Calling display on the MRexperiment object will start a browser
# session with interactive plots.

# require(interactiveDisplay) display(mouseData)
```

6.2 Structural overview

Many studies begin by comparing the abundance composition across sample or feature phenotypes. Often a first step of data analysis is a heatmap, correlation or co-occurence plot or some other data exploratory method. The following functions have been implemented to provide a first step overview of the data:

1. `plotMRheatmap` - heatmap of abundance estimates (Fig. 4 left)
2. `plotCorr` - heatmap of pairwise correlations (Fig. 4 right)
3. `plotOrd` - PCA/CMDS components (Fig. 5 left)
4. `plotRare` - rarefaction effect (Fig. 5 right)
5. `plotBubble` - contingency table style plot (see help)

Each of the above can include phenotypic information in helping to explore the data.

Below we show an example of how to create a heatmap and hierarchical clustering of $\log_2$ transformed counts for the 200 OTUs with the largest overall variance. Red values indicate counts close to zero. Row color labels indicate OTU taxonomic class; column color labels indicate diet (green = high fat, yellow = low fat). Notice the samples cluster by diet in these cases and there are obvious clusters. We then plot a correlation matrix for the same features.

```
trials = pData(mouseData)$diet
heatmapColColors = brewer.pal(12, "Set3")[as.integer(factor(trials))]
heatmapCols = colorRampPalette(brewer.pal(9, "RdBu"))(50)
```

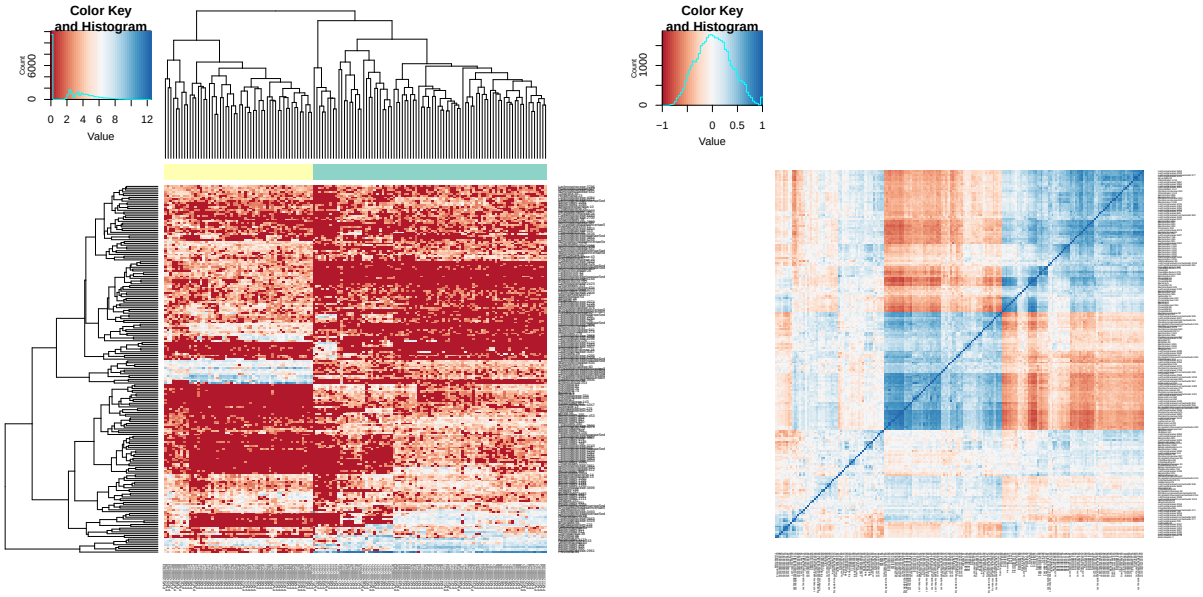

Figure 4: Left) Abundance heatmap (plotMRheatmap). Right) Correlation heatmap (plotCorr).

```
# plotMRheatmap
plotMRheatmap(obj = mouseData, n = 200, cexRow = 0.4, cexCol = 0.4,
    trace = "none", col = heatmapCols, ColSideColors = heatmapColColors)

# plotCorr
plotCorr(obj = mouseData, n = 200, cexRow = 0.25, cexCol = 0.25, trace = "none",
    dendrogram = "none", col = heatmapCols)
```

Below is an example of plotting CMDS plots of the data and the rarefaction effect at the OTU level. None of the data is removed (we recommend removing outliers typically).

```
cl = factor(pData(mouseData)$diet)

# plotOrd - can load vegan and set distfun = vegdist and use
# dist.method='bray'
plotOrd(mouseData, tran = TRUE, usePCA = FALSE, useDist = TRUE, bg = cl,
    pch = 21)

# plotRare
res = plotRare(mouseData, cl = cl, pch = 21, bg = cl)

# Linear fits for plotRare / legend
tmp = lapply(levels(cl), function(lv) lm(res[, "ident"] ~ res[, "libSize"] -
    1, subset = cl == lv))
for (i in 1:length(levels(cl))) {
    abline(tmp[[i]], col = i)
}
legend("topleft", c("Diet 1", "Diet 2"), text.col = c(1, 2), box.col = NA)
```

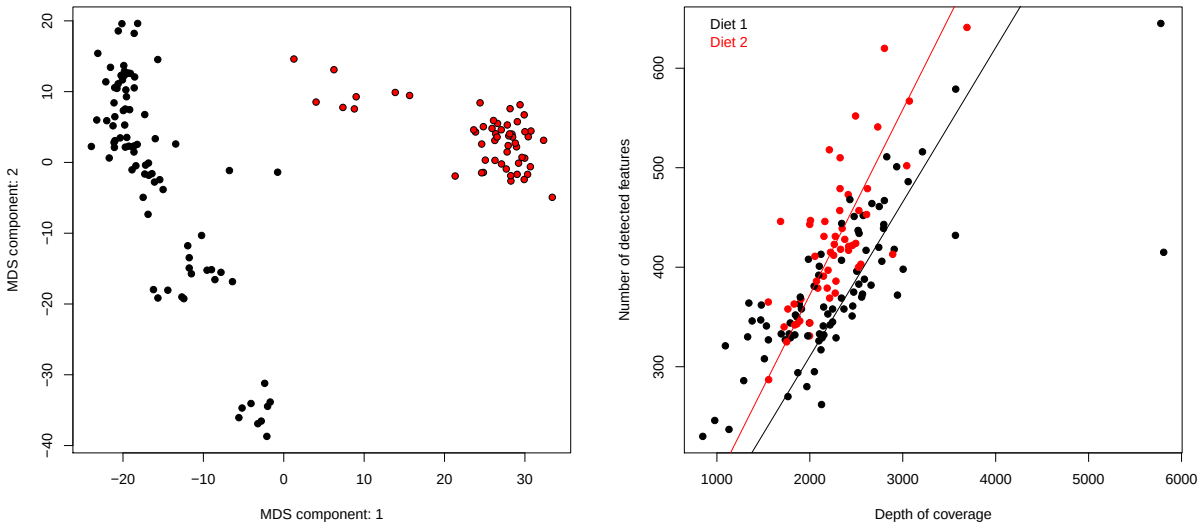

Figure 5: Left) CMDS of features (plotOrd). Right) Rarefaction effect (plotRare).

6.3 Feature specific

Reads clustered with high similarity represent functional or taxonomic units. However, it is possible that reads from the same organism get clustered into multiple OTUs. Following differential abundance analysis. It is important to confirm differential abundance. One way to limit false positives is ensure that the feature is actually abundant (enough positive samples). Another way is to plot the abundances of features similarly annotated.

1. `plotOTU` - abundances of a particular feature by group (Fig. 6 left)
2. `plotGenus` - abundances for several features similarly annotated by group (Fig. 6 right)
3. `plotFeature` - abundances of a particular feature by group (similar to plotOTU, Fig. 7)

Below we use `plotOTU` to plot the normalized log(cpt) of a specific OTU annotated as *Neisseria meningitidis*, in particular the 779th row of lungTrim's count matrix. Using `plotGenus` we plot the normalized log(cpt) of all OTUs annotated as *Neisseria meningitidis*.

It would appear that *Neisseria meningitidis* is differentially more abundant in nonsmokers.

```
head(MRtable(fit, coef = 2, taxa = 1:length(fData(lungTrim)$taxa)))

##     +samples in group 0 +samples in group 1 counts in group 0
## 63                   24                   6              1538
## 779                  23                   7              1512
## 358                  24                   1               390
## 499                  21                   2               326
## 25                   15                  26               162
## 928                   2                  11                 4
##     counts in group 1 smokingStatusSmoker      pvalues
## 63                 11           -4.031612 3.927097e-11
## 779                22           -3.958899 5.751592e-11
```

```
## 358                  1           -2.927686 4.339587e-09
## 499                  2           -2.675306 1.788697e-07
## 25                1893            2.575672 1.360718e-07
## 928                 91            2.574172 3.544957e-04
##       adjPvalues
## 63  2.959194e-08
## 779 2.959194e-08
## 358 8.930871e-07
## 499 1.357269e-05
## 25  1.272890e-05
## 928 1.414122e-03

patients = sapply(strsplit(rownames(pData(lungTrim)), split = "_"),
    function(i) {
        i[3]
    })
pData(lungTrim)$patients = patients
classIndex = list(smoker = which(pData(lungTrim)$SmokingStatus ==
    "Smoker"))
classIndex$nonsmoker = which(pData(lungTrim)$SmokingStatus == "NonSmoker")
otu = 779

# plotOTU
plotOTU(lungTrim, otu = otu, classIndex, main = "Neisseria meningitidis")

# Now multiple OTUs annotated similarly
x = fData(lungTrim)$taxa[otu]
otulist = grep(x, fData(lungTrim)$taxa)

# plotGenus
plotGenus(lungTrim, otulist, classIndex, labs = FALSE, main = "Neisseria meningi

lablist <- c("S", "NS")
axis(1, at = seq(1, 6, by = 1), labels = rep(lablist, times = 3))
```

```
classIndex = list(Western = which(pData(mouseData)$diet == "Western"))
classIndex$BK = which(pData(mouseData)$diet == "BK")
otuIndex = 8770

# par(mfrow=c(1,2))
dates = pData(mouseData)$date
plotFeature(mouseData, norm = FALSE, log = FALSE, otuIndex, classIndex,
    col = dates, sortby = dates, ylab = "Raw reads")
```

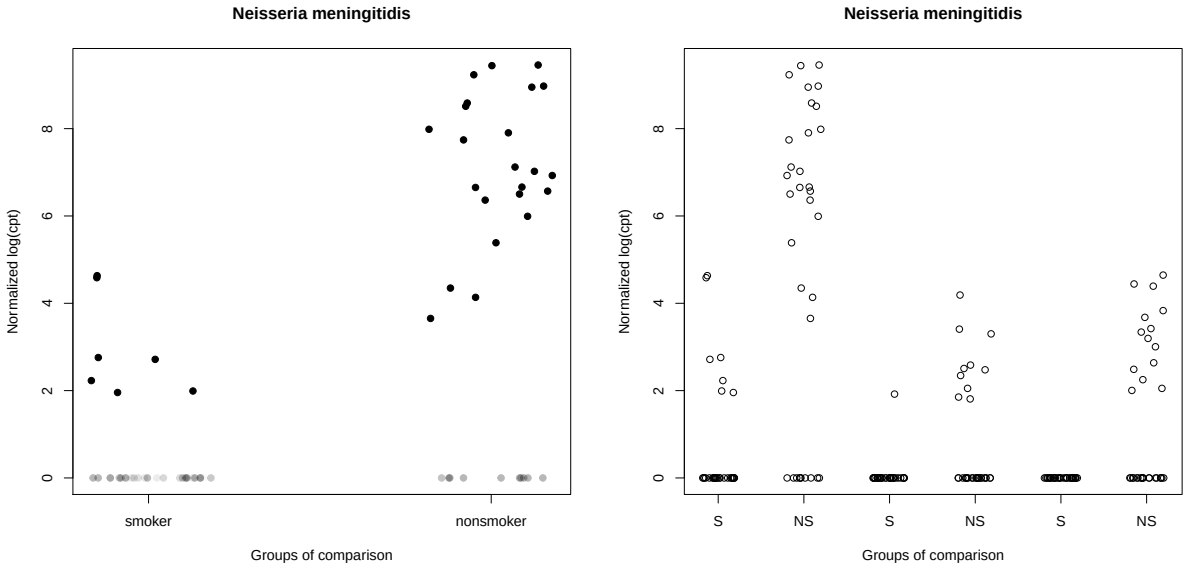

Figure 6: Left) Abundance plot (plotOTU). Right) Multiple OTU abundances (plotGenus).

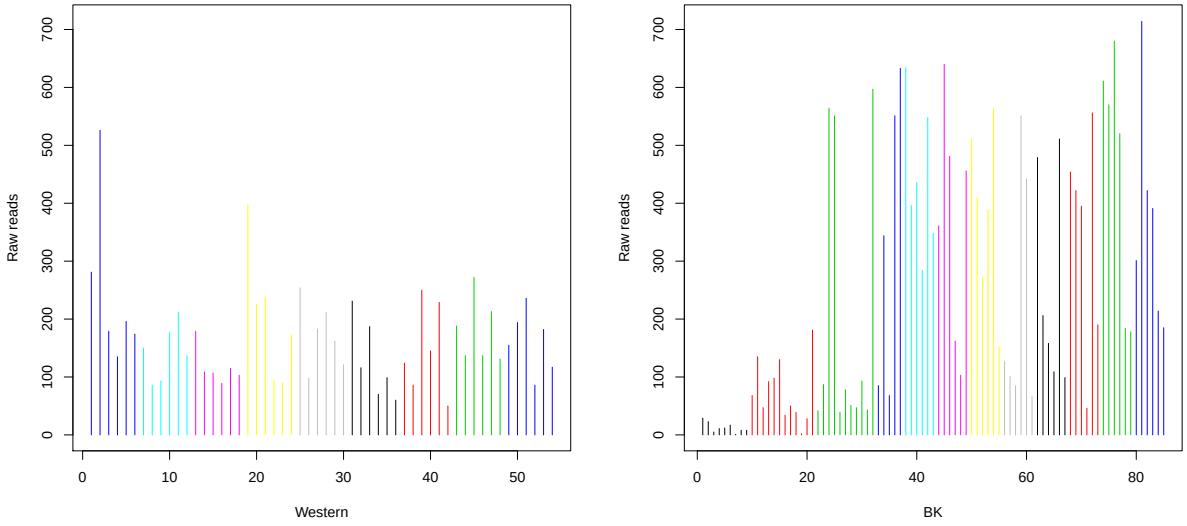

Figure 7: Plot of raw abundances

7 Summary

metagenomeSeq is specifically designed for sparse high-throughput sequencing experiments that addresses the analysis of differential abundance for marker-gene survey data. The package, while designed for marker-gene survey datasets, may be appropriate for other sparse data sets for which the zero-inflated Gaussian mixture model may apply. If you make use of the statistical method please cite our paper. If you made use of the manual/software, please cite the manual/software!

7.1 Citing metagenomeSeq

```
citation("metagenomeSeq")
```

```
##
## Please cite the top for the original statistical method
## and normalization method implemented in metagenomeSeq and
## the bottom for the software/vignette guide. Time series
## analysis/function is described in the third citation.
##
##   JN Paulson, OC Stine, HC Bravo, M Pop.  Differential
##   abundance analysis for microbial marker-gene surveys.
##   Nat Meth Accepted
##
##   JN Paulson, H Talukder, M Pop, HC Bravo. metagenomeSeq:
##   Statistical analysis for sparse high-throughput
##   sequencing. Bioconductor package: 1.12.1.
##   http://cbcb.umd.edu/software/metagenomeSeq
##
##   H Talukder*, JN Paulson*, HC Bravo. Longitudinal
##   differential abundance analysis of marker-gene surveys.
##   Submitted
```

7.2 Session Info

```
sessionInfo()
```

```
## R version 3.2.4 (2016-03-10)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets
## [7] methods   base
##
## other attached packages:
##  [1] gss_2.1-5            metagenomeSeq_1.12.1
##  [3] RColorBrewer_1.1-2   glmnet_2.0-5
```

```
##  [5] foreach_1.4.3        Matrix_1.2-4
##  [7] limma_3.26.9         Biobase_2.30.0
##  [9] BiocGenerics_0.16.1  knitr_1.12.3
##
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5       lattice_0.20-33    stringr_1.0.0
##  [4] highr_0.5.1        caTools_1.17.1     tools_3.2.4
##  [7] grid_3.2.4         KernSmooth_2.23-15 iterators_1.0.8
## [10] gtools_3.5.0       matrixStats_0.50.1 formatR_1.3
## [13] bitops_1.0-6       codetools_0.2-14   evaluate_0.8.3
## [16] gdata_2.17.0       stringi_1.0-1      gplots_3.0.1
```

8 Appendix

8.1 Appendix A: MRexperiment internals

The S4 class system in R allows for object oriented definitions. `metagenomeSeq` makes use of the `Biobase` package in Bioconductor and their virtual-class, `eSet`. Building off of `eSet`, the main S4 class in `metagenomeSeq` is termed `MRexperiment`. `MRexperiment` is a simple extension of `eSet`, adding a single slot, `expSummary`.

The experiment summary slot is a data frame that includes the depth of coverage and the normalization factors for each sample. Future datasets can be formated as MRexperiment objects and analyzed with relative ease. A `MRexperiment` object is created by calling `newMRexperiment`, passing the counts, phenotype and feature data as parameters.

We do not include normalization factors or library size in the currently available slot specified for the sample specific phenotype data. All matrices are organized in the `assayData` slot. All phenotype data (disease status, age, etc.) is stored in `phenoData` and feature data (OTUs, taxanomic assignment to varying levels, etc.) in `featureData`. Additional slots are available for reproducibility and annotation.

8.2 Appendix B: Mathematical model

Defining the class comparison of interest as $k(j) = I\{j \in \text{group}A\}$. The zero-inflated model is defined for the continuity-corrected $\log_2$ of the count data $y_{ij} = \log_2(c_{ij} + 1)$ as a mixture of a point mass at zero $I_{\{0\}}(y_{ij})$ and a count distribution $f_{count}(y_{ij}; \mu_i, \sigma_i^2) \sim N(\mu_i, \sigma_i^2)$. Given mixture parameters π_j, we have that the density of the zero-inflated Gaussian distribution for feature i, in sample j with S_j total counts is:

$$f_{zig}(y_{ij}; \theta) = \pi_j(S_j) \cdot I_{\{0\}}(y_{ij}) + (1 - \pi_j(S_j)) \cdot f_{count}(y_{ij}; \theta) \quad (1)$$

Maximum-likelihood estimates are approximated using an EM algorithm, where we treat mixture membership $\Delta_{ij} = 1$ if y_{ij} is generated from the zero point mass as latent indicator variables [**?**]. We make use of an EM algorithm to account for the linear relationship between sparsity and depth of coverage. The user can specify within the `fitZig` function a non-default zero model that accounts for more than simply the depth of coverage (e.g. country, age, any metadata associated with sparsity, etc.). See Figure 8 for the graphical model.

More information will be included later. For now, please see the online methods in:
http://www.nature.com/nmeth/journal/vaop/ncurrent/full/nmeth.2658.html

8.3 Appendix C: Calculating the proper percentile

To be included: an overview of the two methods implemented for the data driven percentile calculation and more description below.

The choice of the appropriate quantile given is crucial for ensuring that the normalization approach does not introduce normalization-related artifacts in the data. At a high level, the count distribution of samples should all be roughly equivalent and independent of each other up to this quantile under the assumption that, at this range, counts are derived from a common distribution.

More information will be included later. For now, please see the online methods in:
http://www.nature.com/nmeth/journal/vaop/ncurrent/full/nmeth.2658.html

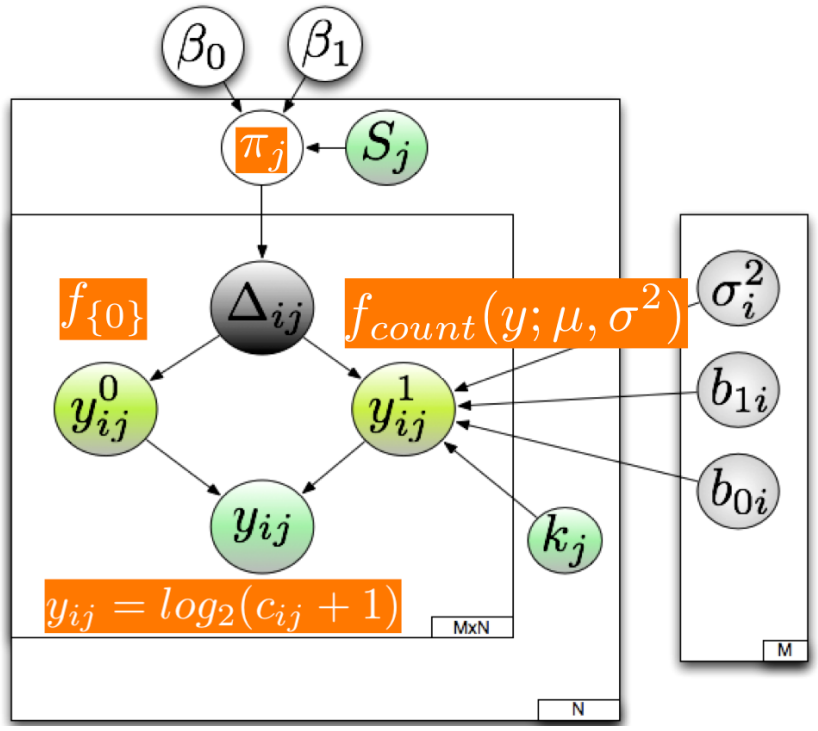

Figure 8: Graphical model. Green nodes represent observed variables: S_j is the total number of reads in sample j; k_j the case-control status of sample j; and y_{ij} the logged normalized counts for feature i in sample j. Yellow nodes represent counts obtained from each mixture component: counts come from either a spike-mass at zero, y_{ij}^0, or the "count" distribution, y_{ij}^1. Grey nodes b_{0i}, b_{1i} and σ_i^2 represent the estimated overall mean, fold-change and variance of the count distribution component for feature i. π_j, is the mixture proportion for sample j which depends on sequencing depth via a linear model defined by parameters β_0 and β_1. The expected value of latent indicator variables Δ_{ij} give the posterior probability of a count being generated from a spike-mass at zero, i.e. y_{ij}^0. We assume M features and N samples.