Vignette

Brody Erlandson

Introduction

In this vignette, we walk through how to use the `FunCZIDM` package and the associated shipy app. We use the case study data in the paper "A Bayesian Functional Concurrent Zero-Inflated Dirichlet Multinomial Regression Model" to provide examples of how you would use and gain inference from the model. This package contains the data from the case study, and the samples from the case study used in the paper. Along with the scripts used to run the model and simulations from the paper. More details on reproducing the results in the paper can be found in the `README.md` file in the packages root directory.

Data Set Up

We can load the data from the case study using `data` function.

```r
library(FunCZIDM)

data(infantData)
```

This loads two objects: `infantCovariates` and `infantCounts`. The first step is to ensure the covariates and counts data are aligned. We can check the two data sets are aligned by checking the subject id's and times are aligned.

```r
# Check count and covariate data are aligned
(all(infantCounts$Subject == infantCovariates$Subject) ||
 all(infantCounts$Day.of.life.sample.obtained
     == infantCovariates$Day.of.life.sample.obtained))
```

```
## [1] TRUE
```

The `FunCZIDM` function requires the categorical covariates to be factors, and will use the reference level as the reference category for the variable. Below, we convert the categorical variables to factors, and ensure the reference level is the one we want.

```r
# Convert the categorical variables to factors
infantCovariates$gender <- as.factor(infantCovariates$gender)
infantCovariates$mode.of.birth <- as.factor(infantCovariates$mode.of.birth)
infantCovariates$Room.category <- as.factor(infantCovariates$Room.category)
infantCovariates$milk <- as.factor(infantCovariates$milk)
infantCovariates$milk <- relevel(infantCovariates$milk, ref="< 10%")
infantCovariates$Period.of.study <- as.factor(infantCovariates$Period.of.study)
infantCovariates$Period.of.study <- relevel(infantCovariates$Period.of.study,
                                            ref="before")

# Check the types of the covariates
str(infantCovariates)
```

```
## 'data.frame':    922 obs. of  9 variables:
```

```
##  $ Subject                         : int  1 1 1 1 1 2 2 2 2 3 ...
##  $ Day.of.life.sample.obtained     : num  54.1 56 59.1 60 67.5 ...
##  $ Gestational.age.at.birth...weeks: num  26 26 26 26 26 26 26 26 26 25 ...
##  $ gender                          : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mode.of.birth                   : Factor w/ 2 levels "caesarian section",..: 1 1 1 1 1 1 1 1 1 1
##  $ Period.of.study                 : Factor w/ 2 levels "before","after": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Room.category                   : Factor w/ 2 levels "open","Single ": 1 1 1 1 1 1 1 1 1 1 ...
##  $ milk                            : Factor w/ 3 levels "< 10%","> 50%",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Proportion.days.on.antibiotics  : num  0.314 0.304 0.288 0.283 0.252 ...
```

The continuous variables will be standardized automatically, and the output of `FunCZIDM` will contain the means and standard deviations used for standardization. Other functions in this package will use this to standardized automatically, so you will always input the original scale and location of the continuous variables.

Now we can remove the subject id's and time variables from the covariates and counts data. The `FunCZIDM` function will use the subject id's and time variables as the `ids` and `varyingCov` arguments, respectively. Lastly, the package requires the counts data to be a matrix.

```r
# getting the index of the subject id's and the time-varying variable
idIdx <- which(colnames(infantCovariates) == "Subject")
timeIdx <- which(colnames(infantCovariates) == "Day.of.life.sample.obtained")
# extracting the subject ids and tv variable
ids <- infantCovariates[, idIdx]
time <- infantCovariates[, timeIdx]
infantCovariates <- infantCovariates[, -c(idIdx, timeIdx), drop = FALSE]

idIdx <- which(colnames(infantCounts) == "Subject")
timeIdx <- which(colnames(infantCounts) == "Day.of.life.sample.obtained")
infantCounts <- infantCounts[, -c(idIdx, timeIdx), drop = FALSE]
infantCounts <- as.matrix(infantCounts)
```

Now we can double check the data is set up correctly.

```r
str(infantCovariates)
```

```
## 'data.frame':    922 obs. of  7 variables:
##  $ Gestational.age.at.birth...weeks: num  26 26 26 26 26 26 26 26 26 25 ...
##  $ gender                          : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mode.of.birth                   : Factor w/ 2 levels "caesarian section",..: 1 1 1 1 1 1 1 1 1 1
##  $ Period.of.study                 : Factor w/ 2 levels "before","after": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Room.category                   : Factor w/ 2 levels "open","Single ": 1 1 1 1 1 1 1 1 1 1 ...
##  $ milk                            : Factor w/ 3 levels "< 10%","> 50%",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Proportion.days.on.antibiotics  : num  0.314 0.304 0.288 0.283 0.252 ...
```

```r
if (is.matrix(infantCounts)) {
  print("Counts data is a matrix.")
}
```

```
## [1] "Counts data is a matrix."
```

```r
colnames(infantCounts)
```

```
##  [1] "Actinobacteria"        "Alphaproteobacteria"   "Bacilli"
##  [4] "Bacteroidia"           "Betaproteobacteria"    "Clostridia"
##  [7] "Cyanobacteria"         "Epsilonproteobacteria" "Erysipelotrichi"
## [10] "Flavobacteria"         "Fusobacteria"          "Gammaproteobacteria"
## [13] "Sphingobacteria"       "Deltaproteobacteria"   "OD1_no_class"
## [16] "unclassified"
```

Running the Model

To run multiple chains, we used a `slurm` array within a high performance computing cluster to sample in parallel. Both the `.R` and `.sh` scripts are provided in `inst/scripts` for ease of reproducibility and implementation.

Once the data is set up, you can sample from the FunC-ZIDM model using the `FunCZIDM` function. A full description of the arguments and options can be found in the reference manual.

```
# This code is not evaluated.
FunCZIDM(infantCounts,
         infantCovariates,
         ids,
         time,
         iter=85000,
         burnIn=75000,
         thin=10,
         toReturn=c("RA"),
         priors=list("first beta sd"=1, "a"=3, "b"=3, "alpha"=.01,
                     "beta"=10, "kappaShape"=100, "kappaRate"=900),
         saveToFile=TRUE,
         saveNullInBetaCI=FALSE)
```

Since we ran multiple processes, we save the output to a file and combine all the chains after each has finished sampling. The `FunCZIDM` package includes the `combineOutputs` function to combine these files, allowing for ease of processing the results. The package includes the combined output from the case study under `inst/extdata`, so it is not necessary to run the sampling.

```
# This code is not evaluated.
outputFiles <- c("output1.rds", "output2.rds", "output3.rds", "output4.rds")
combinedOutput <- combineOutputs(outputFiles, saveToFile=FALSE)
```

Inference

We demonstrate how to use the functions to get inference from the samples using the preloaded samples. The package has the ability to calculate the relative abundance ($\text{RA}_j(t)$) and α-diversity functions ($\alpha(t)$), the change in relative abundance ($\Delta_v \text{RA}_{jp}(t)$) and α-diversity functions ($\Delta_v \alpha_p(t)$), and the functional coefficients ($\beta_{jp}(t)$). Note in the case study script we do not calculate any of these statistics. Instead, we check the `nullInBetaCI.csv` file of the combined output to see which covariates may have meaningful trends, then we use the provided shiny app to generate the figures.

First we load the combined output. The output List will contain many elements, the secondary functions and associated shiny app use these elements. The user will not need many of the elements in the List.

```
combinedOutput <- readRDS(system.file("extdata", "caseStudySamples.rds",
                          package = "FunCZIDM"))
names(combinedOutput)
```

```
##  [1] "interiorKnots"     "boundaryKnots"     "varyingCov"
##  [4] "colMapping"        "XvartoXColMapping" "basisFunc"
##  [7] "df"                "catNames"          "centerScaleList"
## [10] "beta"              "betaAcceptProp"    "rAcceptProp"
## [13] "rMeans"            "etaMeanPropZeros"
```

Relative Abundance

The relative abundance and multiplicative change in relative abundance can be calculated using the `calcRA` and `calcDeltaRA` functions, respectively. First, we need to note the index of each covariate in the model:

```
combinedOutput$colMapping
```

```
## [[1]]
## [1] "(Intercept)"
##
## [[2]]
## [1] "Gestational.age.at.birth...weeks"
##
## [[3]]
## [1] "gendermale"
##
## [[4]]
## [1] "mode.of.birthvaginal delivery"
##
## [[5]]
## [1] "Period.of.studyafter"
##
## [[6]]
## [1] "Room.categorySingle "
##
## [[7]]
## [1] "milk> 50%"
##
## [[8]]
## [1] "milk10-50%"
##
## [[9]]
## [1] "Proportion.days.on.antibiotics"
```

These will be used for filling in the covariate profile and indicating which covariate you want the change in. Similarly, you can view the category names in the order the model uses them with `combinedOutput$catNames`.

```
# always put 1 for the intercept
covProfile <- c(1, 27, 0, 0, 0, 0, 0, 0, .3)
# calculate the relative abundance
RA <- calcRA(combinedOutput, covProfile = covProfile)
dim(RA)
```

```
## [1]  250   16 4000
```

After assigning the desired covariate profile, we get all the categories relative abundance samples. Where we have 250 timepoints, 16 categories, and 4000 samples. Now, we can look into the multiplicative change using the same covariate profile.

```
# calculate the change in relative abundance
change <- c(2, 1, 1) # two week change in gestational age and a 1 for indicating
                  # the change from <10% to >50% breast milk in the diet
forCovs <- c(2, 7, 8) # GA is 2nd idx and >50% BM in diet is 6th idx
forCats <- match(c("Bacilli", "Clostridia", "Gammaproteobacteria"),
                 combinedOutput$catNames) # gets the idx of these taxa
deltaRA <- calcDeltaRA(combinedOutput, change, covProfile = covProfile,
                       forCovs = forCovs, forCats = forCats)
```

```r
names(deltaRA)
```

```
## [1] "Gestational.age.at.birth...weeks" "milk> 50%"
## [3] "milk10-50%"
```

```r
colnames(deltaRA$Gestational.age.at.birth...weeks)
```

```
## [1] "Bacilli"             "Clostridia"          "Gammaproteobacteria"
```

```r
dim(deltaRA$Gestational.age.at.birth...weeks)
```

```
## [1]  250    3 4000
```

We can see the two covariates specified in the List, and for each covariate we have an array with the categories as the columns. The array has 250 timepoints, 3 categories, and 4000 samples.

α-diversity

This works similarly to the relative abundance functions. Using the same covariate profile, we can find the diversity over time:

```r
l <- 0.75 # Hill's diversity parameter
alphaDiv <- calcAlphaDiv(combinedOutput, l=l, covProfile=covProfile)
dim(alphaDiv)
```

```
## [1]  250 4000
```

The diversity uses all the categories, so we just have 250 timepoints and 4000 samples. For the multiplicative change in α-diversity, we only need to specify the covariates of interest and the amount of change.

```r
# calculate the change in relative abundance
change <- c(2, 1) # two week change in gestational age and a 1 for indicating
                  # the change from <10% to >50% breast milk in the diet
forCovs <- c(2, 6) # GA is 2nd idx and >50% BM in diet is 6th idx

deltaAlphaDiv <- calcDeltaAlphaDiv(combinedOutput, change, l=l,
                                   covProfile=covProfile, forCovs=forCovs)
names(deltaAlphaDiv)
```

```
## [1] "Gestational.age.at.birth...weeks" "Room.categorySingle "
```

```r
dim(deltaAlphaDiv$Gestational.age.at.birth...weeks)
```

```
## [1]  250 4000
```

Again, all categories are used in calculating the diversity, so we just have 250 timepoints and 4000 samples.

$\beta_{cp}(t)$

At times, the covariate itself is of interest. These are particularly useful for finding which covariates and taxa combination are of interest. If there are a large number of categories, this helps narrow down the categories to look into.

```r
betas <- getBetaFunctions(combinedOutput)
dim(betas)
```

```
## [1]  250   16 4000
```

This gives us the intercept for each category, and with `cov` you can specify which covariate you want. For example,

```r
betas <- getBetaFunctions(combinedOutput, cov=6)
dim(betas)
```

```
## [1]  250   16 4000
```

gives you the $> 50\%$ breast milk in the diet $\beta_{cp}(t)$.

Generating Datasets

We do not generate datasets in the case study; however, we do in the simulation study. We provide a function to easily generate a dataset if needed for testing or otherwise.

```r
n <- 50 # Number of individuals
c <- 10 # Number of categories
p <- 10 # Number of functional covariates
dataList <- generateData(n, c, p)
```

`dataList` contains the generated data: counts, covariates, ids, timepoints and the true parameters. To recover the true function for each covariate, we use the following code:

```r
cov <- 1 # the first covariate, not the intercept.
cat <- 1
f <- dataList$funcList[[round(dataList$betaFuncMat[cov, cat])]]
plusMinus <- dataList$betaFuncMat[cov, cat] - round(dataList$betaFuncMat[cov, cat])
betaFunc <- ifelse(plusMinus >= 0, 1, -1)*f(dataList$timePoints)
```

This is only for the covariates. The intercept functions can be found in `dataList$betaIntercepts`.

Conclusion

While we covered all the functions in the `FunCZIDM` package, the functionality is not entirely covered. The reference manual contains all of the function parameters, so please refer to the reference for a further description of the functions.

For plotting, the shiny app associated with the package can be used to generate plots with ease. This app should be downloaded locally, then the samples can be uploaded securely.