CCREPE: Compositionality Corrected by Permutation and Renormalization

Emma Schwager, George Weingart, Craig Bielski, Curtis Huttenhower

October 13, 2014

Contents

1 Introduction

ccrepe is a package for analysis of sparse compositional data. Specifically, it determines the significance of association between features in a composition, using any similarity measure (e.g. Pearson correlation, Spearman correlation, etc.) The CCREPE methodology stands for Compositionality Corrected by Renormalization and Permutation, as detailed below. The package also provides a novel similarity measure, the N-dimensional checkerboard score (NC-score), particularly appropriate to compositions derived from microbial community sequencing data. This results in p-values and false discovery rate q-values corrected for the effects of compositionality. The package contains two functions `ccrepe` and `nc.score` and is maintained by the Huttenhower lab (ccrepe-users@googlegroups.com).

2   ccrepe

ccrepe is the main package function. It calculates compositionality-corrected p-values and q-values for a user-selected similarity measure, operating on either one or two input matrices. If given one matrix, all features (columns) in the matrix are compared to each other using the selected similarity measure. If given two matrices, each feature in the first are compared against all features in the second.

2.1   General functionality

Compositional data induces spurious correlations between features due to the nonindependence of values that must sum to a fixed total. CCREPE abrogates this when determining the significance of a similarity measure for each feature pair using two main steps, permutation/renormalization and bootstrapping. First, given two features to compare, CCREPE generates a null distribution of the similarity expected just due to compositionality by iteratively permuting one feature, renormalizing all samples in the composition to their previous sum, and computing the resulting similarity measures. Second, CCREPE bootstraps over sample subsets in order to assess confidence in the "true" similarity measure. Finally, the two resulting distributions are compared using a pooled-variance Z-test to give a compositionality-corrected p-value. False discovery rate q-values are additionally calculated using the Benjamin-Hochberg-Yekutieli procedure. For greater detail, see Faust et al. [2012] and Schwager and Colleagues.

2.2   Arguments

x First *dataframe* or *matrix* containing relative abundances. Columns are features, rows are samples. Rows should therefore sum to a constant. Row names are used for identification if present.

y Second *dataframe* or *matrix* (optional) containing relative abundances. Columns are features, rows are samples. Rows should therefore sum to a constant. If both x and y are specified, they will be merged by row names. If no row names are specified for either or both datasets, the default is to merge by row number.

sim.score Similarity measure, such as cor or nc.score. This can be either an existing R function or user-defined. If the latter, certain properties should be satisfied as detailed below (also see examples). The default similarity measure is Spearman correlation.
A user-defined similarity measure should mimic the interface of cor:
   1. Take either two *vector* inputs one *matrix* or *dataframe* input.
   2. In the case of two inputs, return a single number.
   3. In the case of one input, return a matrix in which the (i,j)th entry is the similarity score for column i and column j in the original matrix.
   4. The resulting matrix (in the case of one input) must be symmetric.
   5. The inputs must be named x and y.

sim.score.args An optional list of arguments for the measurement function. When given, they are passed to the sim.score function directly. For example, in the case of cor, the following would be acceptable:

```
sim.score.args = list(method="spearman", use="complete.obs")
```

Note that this example corresponds to the default behavior.

min.subj Minimum number (count) of samples that must be nonzero in order to apply the similarity measure. This is to ensure that there are sufficient samples to perform a bootstrap (default: 20).

iterations The number of iterations for both bootstrap and permutation calculations (default: 1000).

subset.cols.x Subset of columns from x to work on. The default (NULL) uses all columns. Note that all features are used for normalization, but calculations are performed only with the requested subset.

subset.cols.y Subset of columns from y to work on. The default (NULL) uses all columns. Note that all features are used for normalization, but calculations are performed only with the requested subset.

errthresh1 Maximum allowable probability of getting all zeros in a given bootstrapped column for the first dataset x. If a feature has a number of zeros that makes the probability of obtaining all zeros when sampling with replacement greater than this value, that feature will be excluded from the subsequent analysis. This is to ensure that the standard deviation of the bootstrap sample is non-zero. (default: 0.0001).

verbose If TRUE, print periodic progress of the algorithm through the dataset(s), as well as including more detailed debugging output. (default: FALSE).

iterations.gap If verbose=TRUE, the number of iterations between issuing status messages (default: 100).

distributions Optional output file for detailed log (if given) of all intermediate permutation and renormalization distributions.

compare.within.x A boolean value indicating whether to do comparisons given by taking all subsets of size 2 from subset.cols.x or to do comparisons given by taking all possible combinations of subset.cols.x or subset.cols.y. If TRUE but subset.cols.y=NA, returns all comparisons involving any features in subset.cols.x.

concurrent.output Optional output file to which each comparison will be written as it is calculated.

make.output.table A boolean value indicating whether to include table-formatted output.

2.3 Output

ccrepe returns a *list* containing both the calculation results and the parameters used:

sim.score *matrix* of simliarity scores for all requested comparisons. The (i,j)th element corresponds to the similarity score of column i (or the ith column of subset.cols.1) and column j (or the jth column of subset.cols.1) in one dataset, or to the similarity score of column i (or the ith column of subset.cols.1) in dataset x and column j (or the jth column of subset.cols.2) in dataset y in the case of two datasets.

p.values *matrix* of the corrected p-values for all requested comparisons. The (i,j)th element corresponds to the p-value of the (i,j)th element of sim.score.

q.values *matrix* of the Benjamini-Hochberg-Yekutieli corrected p-values. The (i,j)th element corresponds to the p-value of the (i,j)th element of sim.score.

z.stat *matrix* of the z-statistics used in generating the p-values for all requested comparisons. The (i,j)th element corresponds to the z-statistic generating the (i,j)th element of p.values.

2.4 Usage

```
ccrepe(
 x = NA,
 y = NA,
 sim.score = cor,
 sim.score.args = list(),
 min.subj = 20,
 iterations = 1000,
 subset.cols.x = NULL,
 subset.cols.y = NULL,
 errthresh1 = 1e-04,
```

```
  verbose = FALSE,
  iterations.gap = 100,
  distributions = NA,
  compare.within.x = TRUE,
  concurrent.output = NA,
  make.output.table = FALSE)
```

2.5   Example 1

An example of how to use ccrepe with one dataset.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.rowsum <- apply(data,1,sum)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized
```

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

```
test.input <- data.norm

dimnames(test.input) <- list(c(
 "Sample 1", "Sample 2","Sample 3","Sample 4","Sample 5",
 "Sample 6","Sample 7","Sample 8","Sample 9","Sample 10"),
 c("Feature 1", "Feature 2", "Feature 3","Feature 4"))

test.output <- ccrepe(x=test.input, iterations=20, min.subj=10)
```

```
par(mfrow=c(1,2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
     main="Normalized")
```

```
test.output

## $p.values
##             Feature 1   Feature 2   Feature 3   Feature 4
## Feature 1          NA 0.003169952 0.226610053 0.107535170
## Feature 2 0.003169952          NA 0.002710204 0.566470941
## Feature 3 0.226610053 0.002710204          NA 0.002146112
## Feature 4 0.107535170 0.566470941 0.002146112          NA
##
## $z.stat
##           Feature 1  Feature 2 Feature 3  Feature 4
## Feature 1        NA  2.9507579 -1.209137 -1.6093713
## Feature 2  2.950758         NA -2.998828 -0.5732566
## Feature 3 -1.209137 -2.9988279        NA  3.0692301
## Feature 4 -1.609371 -0.5732566  3.069230         NA
##
## $q.values
##            Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1         NA 0.01501908 0.64420030 0.38212222
## Feature 2 0.01501908         NA 0.01926122 1.34195557
## Feature 3 0.64420030 0.01926122         NA 0.03050452
## Feature 4 0.38212222 1.34195557 0.03050452         NA
```

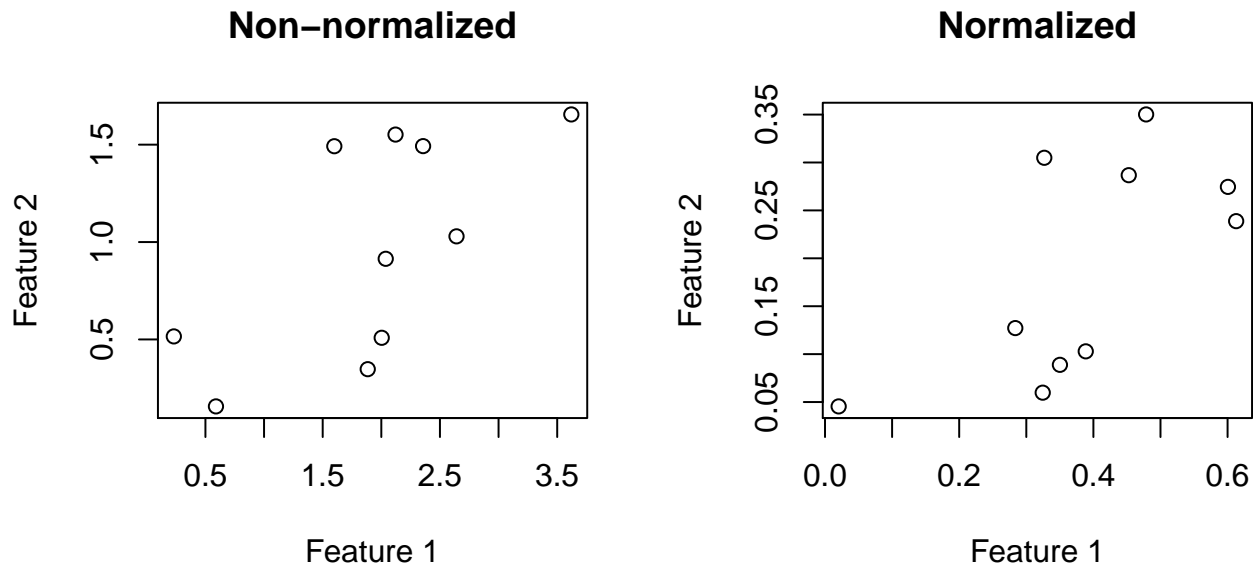**Non–normalized**   **Normalized**

Figure 1: Non-normalized and normalized associations between feature 1 and feature 2. In this case we would expect feature 1 and feature 2 to be associated. In the output we see this by the positive sim.score value in the [1,2] element of test.output$sim.score and the small q-value in the [1,2] element of test.output$q.values.

```
##
## $sim.score
##             Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1         NA  0.5878788 -0.7696970 -0.8666667
## Feature 2  0.5878788         NA -0.8303030 -0.5393939
## Feature 3 -0.7696970 -0.8303030         NA  0.6606061
## Feature 4 -0.8666667 -0.5393939  0.6606061         NA
```

2.6   Example 2

An example of how to use ccrepe with two datasets.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.rowsum <- apply(data,1,sum)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

## [1] 1 1 1 1 1 1 1 1 1 1

test.input <- data.norm


data2 <- matrix(rlnorm(105,meanlog=0,sdlog=1),nrow=15,ncol=7)
aligned.rows <- c(seq(1,4),seq(6,9),11,12)  # The datasets dont need
                                            # to have subjects line up exactly
data2[aligned.rows,1] <-  2*data[,3] + rnorm(10,0,1)
data2.rowsum <- apply(data2,1,sum)
```

```
data2.norm <- data2/data2.rowsum
apply(data2.norm,1,sum)  # The rows sum to 1, so the data are normalized

##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

test.input.2 <- data2.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))
dimnames(test.input.2) <- list(paste("Sample",c(seq(1,4),11,seq(5,8),12,9,10,13,14,15)),paste("Feature",se

test.output.two.datasets <- ccrepe(x=test.input, y=test.input.2, iterations=20, min.subj=10)

par(mfrow=c(1,2))
plot(data2[aligned.rows,1],data[,3],xlab="dataset 2: Feature 1",ylab="dataset 1: Feature 3",main="Non-norm
plot(data2.norm[aligned.rows,1],data.norm[,3],xlab="dataset 2: Feature 1",ylab="dataset 1: Feature 3",
     main="Normalized")
```
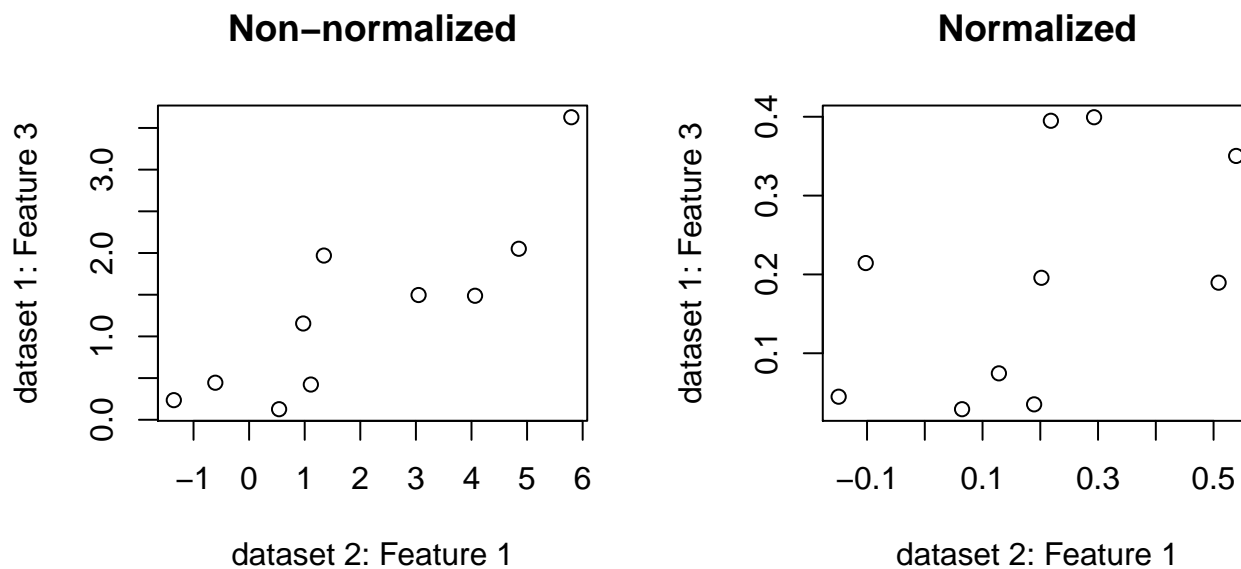

Figure 2: Non-normalized and normalized associations between feature 1 and feature 2. In this case we would expect feature 1 and feature 2 to be associated. In the output we see this by the positive sim.score value in the [1,2] element of test.output$sim.score and the small q-value in the [1,2] element of test.output$q.values.

```
test.output.two.datasets

## $p.values
##           Feature 1 Feature 2 Feature 3 Feature 4 Feature 5  Feature 6
## Feature 1 0.58469169 0.7478253 0.4093648 0.7198538 0.5963036 0.26514166
## Feature 2 0.97734862 0.5604220 0.7561699 0.7279331 0.8243240 0.09691887
## Feature 3 0.03457125 0.3678019 0.5994461 0.7197196 0.2162972 0.33960011
## Feature 4 0.26919276 0.6951659 0.6103346 0.7287906 0.3600630 0.59836747
##            Feature 7
## Feature 1 0.2614908
## Feature 2 0.6077187
## Feature 3 0.6678251
## Feature 4 0.4907975
```

```
##
## $z.stat
##           Feature 1  Feature 2  Feature 3  Feature 4  Feature 5  Feature 6
## Feature 1  0.5465445  0.3215082 -0.8250120 -0.3586543 -0.5297236  1.1143206
## Feature 2 -0.0283931 -0.5822149 -0.3105143  0.3478763  0.2219870  1.6599780
## Feature 3  2.1133450 -0.9005984  0.5251972 -0.3588336 -1.2364342 -0.9549557
## Feature 4 -1.1049234  0.3918543  0.5095958  0.3467347  0.9152451 -0.5267496
##           Feature 7
## Feature 1 -1.1228745
## Feature 2 -0.5133325
## Feature 3  0.4291349
## Feature 4  0.6890410
##
## $q.values
##           Feature 1 Feature 2 Feature 3 Feature 4 Feature 5 Feature 6
## Feature 1  4.923273  3.274391  4.481061  3.581723  4.662403  5.804681
## Feature 2  3.820866  5.112158  3.183585  3.464448  3.341986  5.304552
## Feature 3  3.784299  4.473442  4.101102  3.751582  7.892236  5.310558
## Feature 4  4.911142  3.804774  3.711640  3.324007  4.926731  4.366637
##           Feature 7
## Feature 1  7.155942
## Feature 2  3.913128
## Feature 3  3.847508
## Feature 4  4.884049
##
## $sim.score
##           Feature 1   Feature 2    Feature 3    Feature 4    Feature 5
## Feature 1  0.1878788  0.09090909 -0.296969697 -0.309090909 -0.04242424
## Feature 2  0.1030303 -0.33333333  0.006060606 -0.006060606  0.01818182
## Feature 3  0.5757576 -0.34545455  0.115151515  0.054545455 -0.38181818
## Feature 4 -0.4909091  0.21212121  0.200000000  0.175757576  0.24848485
##             Feature 6  Feature 7
## Feature 1  0.09090909 -0.2848485
## Feature 2  0.56363636 -0.1393939
## Feature 3 -0.29696970  0.2969697
## Feature 4 -0.10303030  0.1030303
```

2.7   Example 3

An example of how to use ccrepe with nc.score as the similarity score.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.rowsum <- apply(data,1,sum)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

##  [1] 1 1 1 1 1 1 1 1 1 1

test.input <- data.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))

test.output.nc.score    <- ccrepe(x=test.input, sim.score=nc.score, iterations=20, min.subj=10)
```

```
par(mfrow=c(1,2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
    main="Normalized")
```
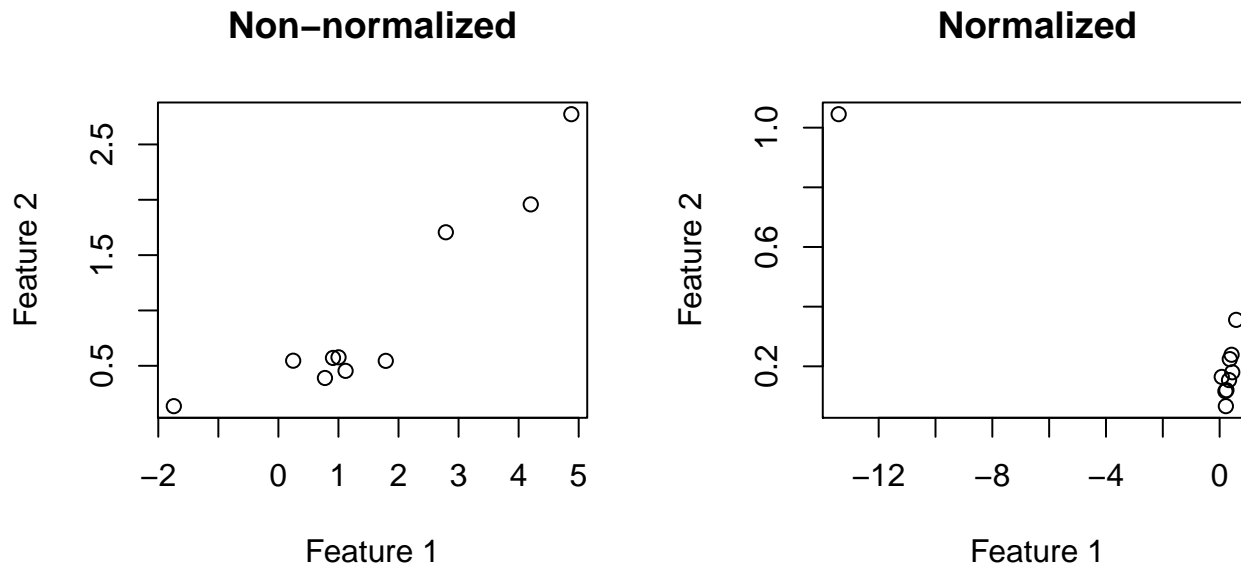

Figure 3: Non-normalized and normalized associations between feature 1 and feature 2. In this case we would expect feature 1 and feature 2 to be associated. In the output we see this by the positive sim.score value in the [1,2] element of test.output$sim.score and the small q-value in the [1,2] element of test.output$q.values. In this case, however, the sim.score represents the NC-Score between two features rather than the Spearman correlation.

```
test.output.nc.score

## $p.values
##          Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1       NA 0.1152068 0.3139179 0.7153109
## Feature 2 0.1152068       NA 0.1203355 0.5853376
## Feature 3 0.3139179 0.1203355       NA 0.5761035
## Feature 4 0.7153109 0.5853376 0.5761035       NA
##
## $z.stat
##           Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1        NA  1.5752152 -1.0070352 -0.3647328
## Feature 2 1.5752152         NA -1.5533669  0.5456049
## Feature 3 -1.0070352 -1.5533669        NA  0.5590854
## Feature 4 -0.3647328  0.5456049  0.5590854        NA
##
## $q.values
##          Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1       NA 1.6375318 1.4873272  1.694554
## Feature 2 1.637532        NA 0.8552155  1.663980
## Feature 3 1.487327 0.8552155        NA  2.047162
## Feature 4 1.694554 1.6639802 2.0471622        NA
```

```
##
## $sim.score
##            Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1        NA  0.3809524 -0.5887446 -0.1991342
## Feature 2  0.3809524         NA -0.2857143 -0.0952381
## Feature 3 -0.5887446 -0.2857143         NA -0.0952381
## Feature 4 -0.1991342 -0.0952381 -0.0952381         NA
```

2.8   Example 4

An example of how to use `ccrepe` with a user-defined `sim.score` function.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.rowsum <- apply(data,1,sum)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

##  [1] 1 1 1 1 1 1 1 1 1 1

test.input <- data.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))

my.test.sim.score <- function(x,y=NA,constant=0.5){
      if(is.vector(x) && is.vector(y)) return(constant)
        if(is.matrix(x) && is.na(y)) return(matrix(rep(constant,ncol(x)^2),ncol=ncol(x)))
        if(is.data.frame(x) && is.na(y)) return(matrix(rep(constant,ncol(x)^2),ncol=ncol(x)))
        else stop('ERROR')
  }

test.output.sim.score    <- ccrepe(x=test.input, sim.score=my.test.sim.score, iterations=20, min.subj=10,

par(mfrow=c(1,2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
    main="Normalized")

test.output.sim.score

## $p.values
##            Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1        NA       NaN       NaN       NaN
## Feature 2       NaN        NA       NaN       NaN
## Feature 3       NaN       NaN        NA       NaN
## Feature 4       NaN       NaN       NaN        NA
##
## $z.stat
##            Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1        NA       NaN       NaN       NaN
## Feature 2       NaN        NA       NaN       NaN
## Feature 3       NaN       NaN        NA       NaN
## Feature 4       NaN       NaN       NaN        NA
##
## $q.values
```
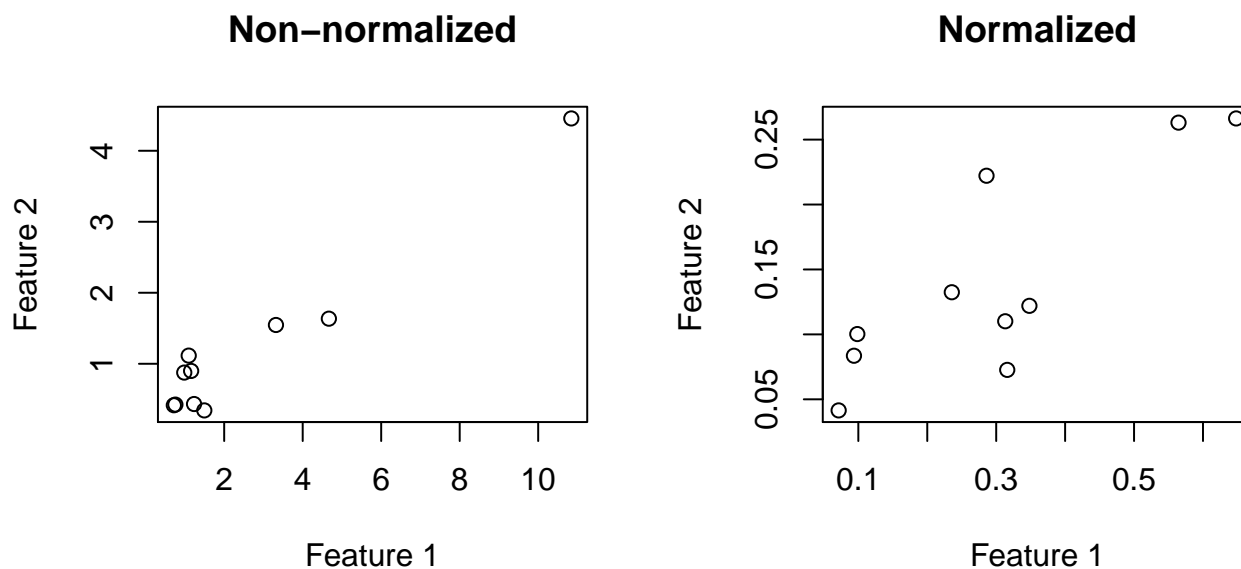
Figure 4: Non-normalized and normalized associations between feature 1 and feature 2. In this case we would expect feature 1 and feature 2 to be associated. Note that the values of sim.score are all 0.6 and none of the p-values are very small because of the arbitrary definition of the similarity score.

```
##           Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1       NA       NaN       NaN       NaN
## Feature 2      NaN        NA       NaN       NaN
## Feature 3      NaN       NaN        NA       NaN
## Feature 4      NaN       NaN       NaN        NA
##
## $sim.score
##           Feature 1 Feature 2 Feature 3 Feature 4
## Feature 1       NA       0.6       0.6       0.6
## Feature 2      0.6        NA       0.6       0.6
## Feature 3      0.6       0.6        NA       0.6
## Feature 4      0.6       0.6       0.6        NA
```

2.9   Example 5

An example of how to use `ccrepe` when specifying column subsets.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data.rowsum <- apply(data,1,sum)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

## [1] 1 1 1 1 1 1 1 1 1 1

test.input <- data.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))
```

```
test.output.1.3      <- ccrepe(x=test.input, iterations=20, min.subj=10, subset.cols.x=c(1,3))
test.output.1        <- ccrepe(x=test.input, iterations=20, min.subj=10, subset.cols.x=c(1), compare.within
test.output.12.3     <- ccrepe(x=test.input, iterations=20, min.subj=10, subset.cols.x=c(1,2),subset.cols.y=
test.output.1.3$sim.score

##           Feature 1 Feature 2  Feature 3 Feature 4
## Feature 1        NA        NA -0.4787879        NA
## Feature 2        NA        NA         NA        NA
## Feature 3 -0.4787879        NA         NA        NA
## Feature 4        NA        NA         NA        NA

test.output.1$sim.score

##           Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1        NA -0.3575758 -0.4787879 -0.4666667
## Feature 2 -0.3575758        NA         NA        NA
## Feature 3 -0.4787879        NA         NA        NA
## Feature 4 -0.4666667        NA         NA        NA

test.output.12.3$sim.score

##           Feature 1   Feature 2   Feature 3 Feature 4
## Feature 1        NA         NA -0.47878788        NA
## Feature 2        NA         NA -0.04242424        NA
## Feature 3 -0.4787879 -0.04242424         NA        NA
## Feature 4        NA         NA         NA        NA
```

3   nc.score

The `nc.score` similarity measure is an N-dimensional extension of the checkerboard score particularly suited to similarity score calculations between compositions derived from ecological relative abundance measurements. In such cases, features typically represent species abundances, and the NC-score discretizes these continuous values into one of N bins before computing a normalized similarity of co-occurrence or co-exclusion. This can be used as a standalone function or with `ccrepe` as above to obtain compositionality-corrected p-values.

3.1   General Functionality

The NC-score is an extension to Diamond's checkerboard score (see Cody and Diamond [1975]) to ordinal data. The generalization of the checkerboard score is through defining general co-variation and co-exclusion patterns in ordinal data. With ordinal data, the checkerboard considers the 2×2 submatrices which are of the form:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

In the cases that $a < b$ and $c < d$ (or $a > b$ and $c > d$), the submatrix is considered a co-variation pattern; conversely, if $a > b$ and $c < d$ (or $a < b$ and $c > d$) then the submatrix is considered a co-exclusion pattern. The number of co-exclusion patterns for n bins is adjusted based on the expected ratio of co-variaion to co-exclusion patterns in any given sample. The adjustment factor is given by:

$$1.5 * \frac{n(n-1)}{n^2 - n + 1},$$

with the complete derivation being left to Bielski [2013].

The function as implemented here first performs basic quality control filtering of input relative abundance data. It then transforms relative abundances to ordinal values based on user-provided or default bin thresholds. It then computes a raw NC-score:

$$C_{ij} - 1.5 * \frac{n(n-1)}{n^2 - n + 1} * D_{ij}$$

where C_{ij} and D_{ij} are the total number of co-variation and co-exclusion patterns, respectively, between species i and j and where n is the number of bins.

The raw NC-score is then normalized to between -1 and 1 for n bins and s samples by dividing by the maximum possible NC-score, which is based on the maximum number of co-variation patterns possible. The normalized NC-score is then analogous to Pearsons ρ, where positive values indicate more co-variation than co-exclusion patterns and the magnitude of the score indicates the strength of the association between the species. The derivation of this normalization factor can be found in Bielski [2013]. The normalization factor is given by:

$$\binom{s}{2} - \left[s \bmod n * \binom{\lfloor \frac{s}{n} \rfloor + 1}{2} + (n - s \bmod n) * \binom{\lfloor \frac{s}{n} \rfloor}{2} \right].$$

3.2   Arguments

x First numerical *vector*, or single *dataframe* or *matrix*, containing relative abundances. If the latter, columns are features, rows are samples. Rows should therefore sum to a constant.

y If provided, second numerical *vector* containing relative abundances. If given, x must be a *vector* as well.

bins Either the number of bins to use or a *vector* specifying bin edges. If a single number is given, this is used as the number of bins with the `discretize` function of the package *infotheo*. If a *vector* is specified, the function `findInterval` is used to discretize the data. The default behavior is to use the defaults for the `discretize` function.

verbose Request verbose output.

min.abundance Minimum abundance threshold for quality control filtering. For a feature to be included, it must take a value of at least `min.abundance` in at least `min.samples` percent of samples.

min.samples Minimum sample threshold for quality control filtering. For a feature to be included, it must take a value of at least `min.abundance` in at least `min.samples` percent of samples.

3.3   Output

`nc.score` returns either a single number (if called with two vectors) or a *matrix* of all pairwise scores (if called with a *matrix*) of normalized scores.

3.4   Usage

```
nc.score(
 x = NA,
 y = NA,
 bins = NA,
 verbose = FALSE,
 min.abundance = 1e-04,
 min.samples = 0.1)
```

3.5 Example 1

An example of using `nc.score` to get a single similarity score or a matrix.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data.rowsum <- apply(data,1,sum)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

##  [1] 0.7017173 1.4021388 0.9308655 0.6099699 0.9405957 2.0834699 1.5411985
##  [8] 2.0631983 1.5918231 0.8070520

test.input <- data.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))

test.output.matrix <- nc.score(x=test.input)
test.output.num    <- nc.score(x=test.input[,1],y=test.input[,2])
```

```
par(mfrow=c(1, 2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
    main="Normalized")
```
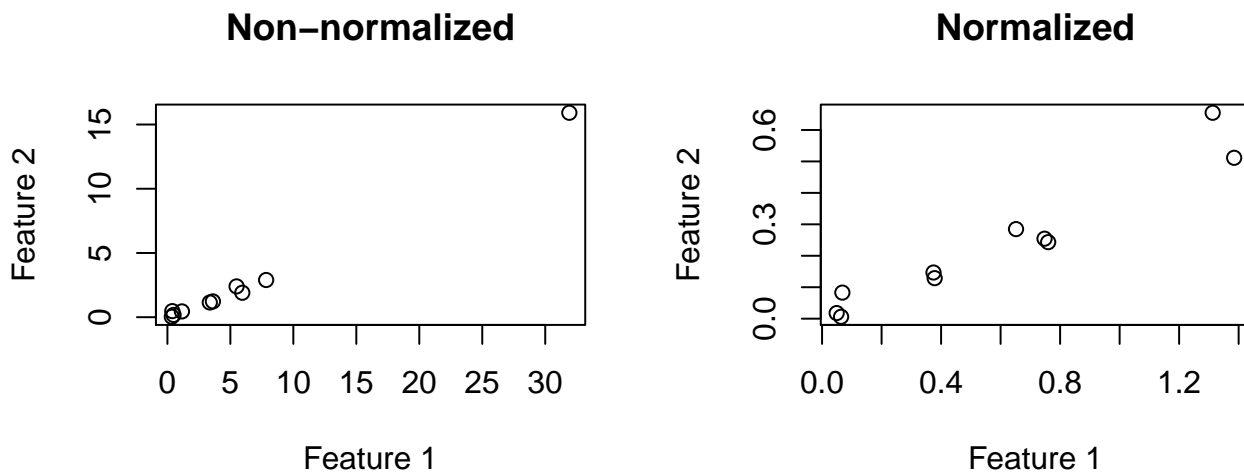

Figure 5: Non-normalized and normalized associations between feature 1 and feature 2 of the second example. Again, we expect to observe a positive association between feature 1 and feature 2. In terms of generalized checkerboard scores, we would expect to see more co-variation patterns than co-exclusion patterns. This is shown by the positive and relatively high value of the [1,2] element of test.output.matrix (which is identical to test.output.num)

```
test.output.matrix

##            Feature 1  Feature 2  Feature 3  Feature 4
## Feature 1         NA  0.6190476 -0.2857143 -0.3333333
## Feature 2  0.6190476         NA -0.3116883 -0.0952381
## Feature 3 -0.2857143 -0.3116883         NA -0.0952381
## Feature 4 -0.3333333 -0.0952381 -0.0952381         NA
```

```
test.output.num
```

```
## [1] 0.6190476
```

3.6 Example 2

An example of using `nc.score` with an aribitrary bin number.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data.rowsum <- apply(data,1,sum)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized
```

```
##  [1] 0.7626933 1.6958818 1.9893117 0.9187088 0.2392500 1.4145728 2.2395308
##  [8] 0.7142748 0.7072317 1.4199046
```

```
test.input <- data.norm
```

```
dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))
```

```
test.output <- nc.score(x=test.input,bins=2)
```

```
par(mfrow=c(1, 2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
     main="Normalized")
```
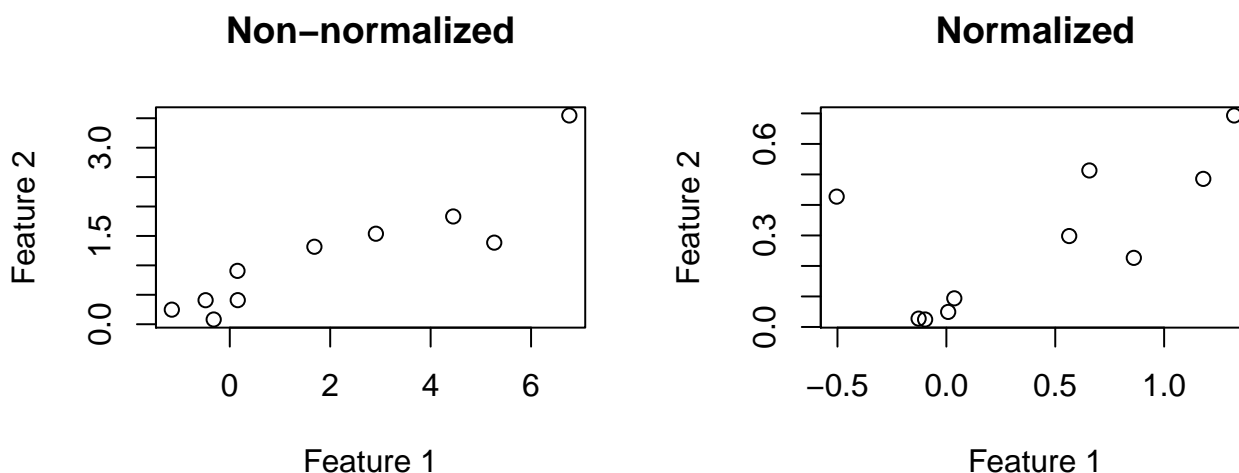

Figure 6: Non-normalized and normalized associations between feature 1 and feature 2 of the second example. Again, we expect to observe a positive association between feature 1 and feature 2. In terms of generalized checkerboard scores, we would expect to see more co-variation patterns than co-exclusion patterns. This is shown by the positive and relatively high value in the [1,2] element of test.output. In this case, the smaller bin number yields a smaller NC-score because of the coarser partitioning of the data.

```
test.output
```

```
##          Feature 1 Feature 2 Feature 3 Feature 4
```

```
## Feature 1          NA          0.6        -0.2        -0.2
## Feature 2         0.6           NA         0.2        -0.6
## Feature 3        -0.2          0.2          NA        -0.6
## Feature 4        -0.2         -0.6        -0.6          NA
```

3.7   Example 3

An example of using `nc.score` with user-defined bin edges.

```
data <- matrix(rlnorm(40,meanlog=0,sdlog=1),nrow=10,ncol=4)
data.rowsum <- apply(data,1,sum)
data[,1] = 2*data[,2] + rnorm(10,0,1)
data.norm <- data/data.rowsum
apply(data.norm,1,sum)  # The rows sum to 1, so the data are normalized

##  [1] 0.6158455 1.0632493 0.8725077 2.2238021 1.0366282 0.6856551 1.3713757
##  [8] 1.3869975 0.4971866 0.7765816

test.input <- data.norm

dimnames(test.input) <- list(paste("Sample",seq(1,10)),paste("Feature",seq(1,4)))

test.output <- nc.score(x=test.input,bins = c(0.001,0.1,0.25,0.6))

par(mfrow=c(1, 2))
plot(data[,1],data[,2],xlab="Feature 1",ylab="Feature 2",main="Non-normalized")
plot(data.norm[,1],data.norm[,2],xlab="Feature 1",ylab="Feature 2",
     main="Normalized")
```
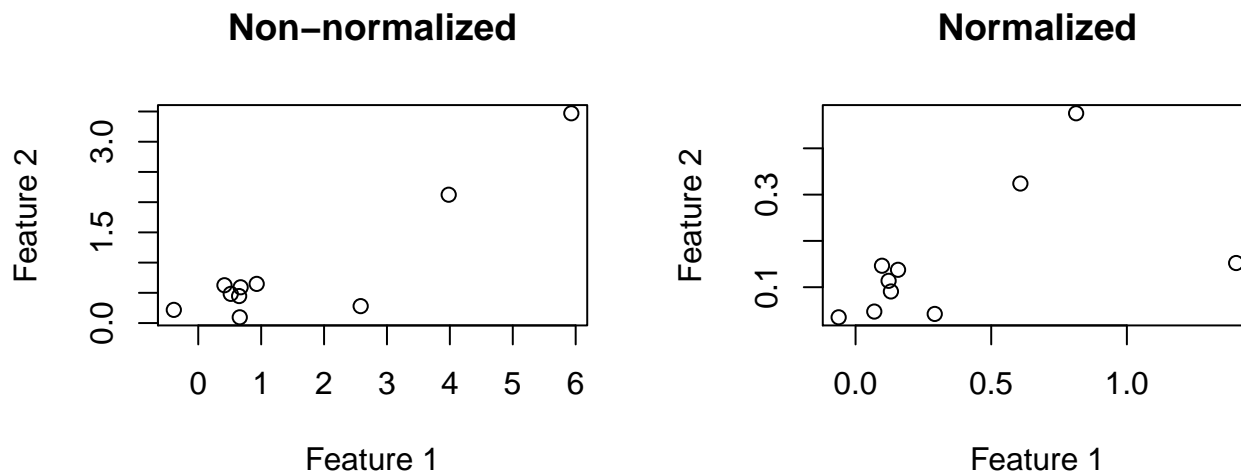

Figure 7: Non-normalized and normalized associations between feature 1 and feature 2 of the second example. Again, we expect to observe a positive association between feature 1 and feature 2. In terms of generalized checkerboard scores, we would expect to see more co-variation patterns than co-exclusion patterns. This is shown by the positive and relatively high value in the [1,2] element of test.output. The bin edges specified here represent almost absent ([0,0.001)), low abundance ([0.001,0.1)), medium abundance ([0.1,0.25)), and high abundance ([0.6,1)).

```
test.output

##            Feature 1  Feature 2   Feature 3  Feature 4
## Feature 1        NA  0.5035714 -0.05357143 -0.3250000
## Feature 2 0.50357143        NA -0.26403326 -0.4324324
## Feature 3 -0.05357143 -0.2640333         NA  0.2453222
## Feature 4 -0.32500000 -0.4324324  0.24532225        NA
```

4   References

References

Craig Bielski. Extending the Checkerboard Score to Ordinal Data: A Methodology for Detecting Species-Level Co-Variation and Co-Exclusion Patterns in the Human Microbiome. Master's thesis, Harvard School of Public Health, Boston, USA, 2013.

Martin Leonard Cody and Jared Mason Diamond. *Ecology and evolution of communities*. Harvard University Press, 1975.

Karoline Faust, J Fah Sathirapongsasuti, Jacques Izard, Nicola Segata, Dirk Gevers, Jeroen Raes, and Curtis Huttenhower. Microbial co-occurrence relationships in the human microbiome. *PLoS computational biology*, 8(7):e1002606, 2012.

Emma Schwager and Colleagues. Detecting statistically significant associtions between sparse and high dimensional compositioanl data. In Progress.