EBIO 5460

Supplementary Exercise – Variation and Dissimilarity Partitioning

Variation Partitioning

Biologists often have multiple sets of explanatory variables that potentially explain variation in response variables. When this is the case, it can be useful to partition the variation in a response variable that is due to one set of explanatory variables from that due to other sets of explanatory variables. For multivariate data, this can be accomplished by comparing constrained ordinations that have different explanatory datasets. The vegan function for this method is "varpart". This function computes RDAs for different explanatory datasets separately and combined, and then computes adjusted R^2 values for fractions of the total variation that is explained by each explanatory dataset and their combined influence.

To demonstrate this technique, let's open the vegan package and examine the different kinds of partitioning that can be accomplished with varpart:

```
library(vegan)
showvarparts(2)
showvarparts(3)
showvarparts(4)
```

The function can partition variation due to 2, 3 or 4 explanatory datasets. With 2 explanatory datasets, called X1 and X2, varpart partitions the variation due to X1 into fractions 'a' and 'b' and the variation due to X2 into fractions 'b' and 'c'. The fraction 'b' is the variation that is due to both X1 and X2 and cannot be distinguished between the two explanatory datasets. The remaining variation that is not due to the explanatory variables is called 'd', which represents the residual variation. Variation partitioning with 3 or 4 explanatory datasets operates in the same way, but the fractions (a, b, c, etc.) are given different names than with 2 explanatory datasets.

Let's give this a try with the dune data. First, load the data and calculate Bray-Curtis dissimilarities.

```
data(dune)
data(dune.env)
dune.rel <- decostand(dune, "total")
dune.bc <- vegdist(dune.rel, method="bray")
```

The function uses distance-based RDA (db-RDA) to partition variation, which means that it can accommodate any dissimilarity or distance measure. Now, let's apply the varpart function and call up the results. Here, we are telling the function to analyze dissimilarities as a function of land management (X1) and as a function of soil properties (X2).

```
mod <- varpart(dune.bc, ~Management, ~A1+Moisture, data=dune.env)
mod
```

The model output shows the adjusted R² values for each individual fraction and for combinations of fractions corresponding to component db-RDAs. To visualize the individual fractions, we can use the plot function.

```
plot(mod)
```

The varpart function computes the individual fractions by comparing adjusted R² values for the different component db-RDAs. In this case, there are three component db-RDAs corresponding to X1 (fractions a+b), X2 (fractions b+c) and X1+X2 (fractions a+b+c). To obtain the individual fraction 'a', the function subtracted the adjusted R² value of the db-RDA with X2 from the adjusted R² value of the db-RDA with X1+X2.

This calculation can be shown by comparing the output from the individual db-RDAs:

```
ab <- dbrda(dune.bc ~ Management, dune.env)
r2.ab <- RsquareAdj(ab)
bc <- dbrda(dune.bc ~ A1+Moisture, dune.env)
r2.bc <- RsquareAdj(bc)
abc <- dbrda(dune.bc ~ Management+A1+Moisture, dune.env)
r2.abc <- RsquareAdj(abc)

a <- r2.abc$adj.r.squared - r2.bc$adj.r.squared
a
c <- r2.abc$adj.r.squared - r2.ab$adj.r.squared
c
```

The individual fractions 'a' and 'c' can also be calculated with db-RDAs that evaluate the contribution of X1 while accounting for X2 using the 'Condition' command.

```
a.alt <- dbrda(dune.bc ~ Management + Condition(A1+Moisture), dune.env)
RsquareAdj(a.alt)
c.alt <- dbrda(dune.bc ~ A1+Moisture + Condition(Management), dune.env)
RsquareAdj(c.alt)
```

The significance of the fractions can be evaluated with permutation tests, such as:

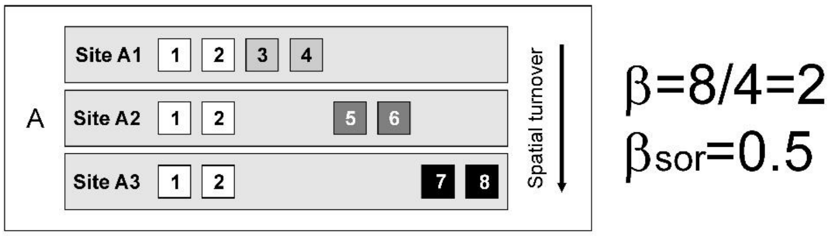
```
anova.cca(a)
anova.cca(c)
anova.cca(ab)
anova.cca(bc)
anova.cca(abc)
```

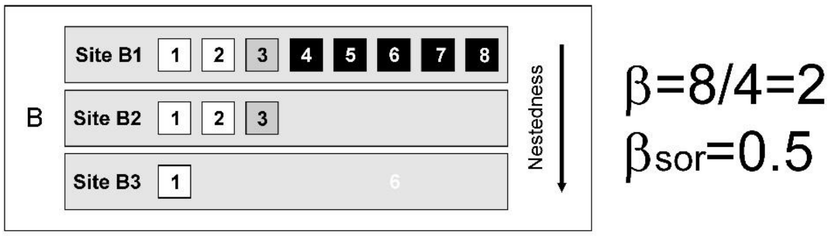
Notice that we did not specify models or evaluate the significance of the 'b' and 'd' fractions of the total variation. We did this because these fractions cannot be computed directly and need to be calculated from the other fractions. These fractions are untestable because they are calculated indirectly.

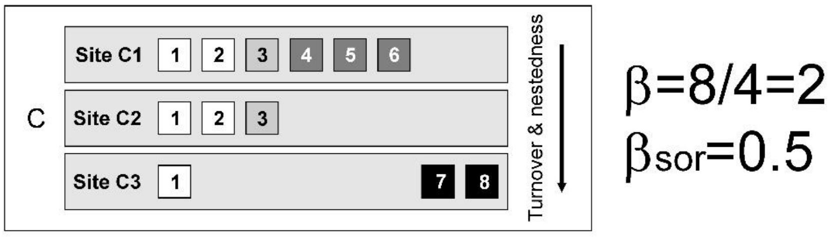
Dissimilarity Partitioning

Most of this semester, we have evaluated abundance-based measures of dissimilarity and have focused on species turnover by calculating relative species abundances, which controls for total abundances across all species. However, this cannot be accomplished for presence/absence data because we do not have a measure of total abundance. Luckily, there is a partitioning procedure that can distinguish the dissimilarity due to turnover and that due to nestedness (ie. total abundances) from total dissimilarity.

We talked about spatial turnover and nestedness in our first lecture and further information can be found at https://methodsblog.wordpress.com/2015/05/27/beta_diversity/. Below are several figures borrowed from this website, which show how different combinations of turnover and nestedness can result in the same values for beta diversity (gamma/average alpha) and related dissimilarity indices (Sorensen).







The above situations are problematic because we do not know how much of beta diversity can be attributed to turnover and/or nestedness in the data. To solve this problem, the R package ‘betapart’ was designed to partition these components of total dissimilarity. Please download and then open the betapart package.

```
library(betapart)
```

Now let’s give the package a try with the dune data transformed into presence/absence data.

```
dune.pa <- decostand(dune, "pa")
```

The partitioning function in this package is called ‘beta.multi’, which calculates the total dissimilarity across all sites and the components of the total dissimilarity that are due to turnover and nestedness. There are two options for dissimilarities, Sorensen and Jaccard, which is specified with the index.family command within the function.

```
multi.dune.sor = beta.multi(dune.pa, index.family="sorensen")
multi.dune.sor

multi.dune.jac = beta.multi(dune.pa, index.family="jaccard")
multi.dune.jac
```

Three values are provided in the output, corresponding to total dissimilarity (SOR or JAC), dissimilarity due to turnover (SIM or JTU), and dissimilarity due to nestedness (SNE or JNE).

Instead of across all sites together, we might also want to partition these components of dissimilarity for each pair of sites. This can be done with the ‘beta.pair’ function. Below, we compute 3 matrices corresponding to total dissimilarities, dissimilarities due to turnover, and dissimilarities due to nestedness. Then we save each matrix as a separate object.

```
pairs.dune.pa = beta.pair(dune.pa, index.family="sorensen")
dune.pa.sim = pairs.dune.pa$beta.sim
dune.pa.sne = pairs.dune.pa$beta.sne
dune.pa.sor = pairs.dune.pa$beta.sor
```

These matrices are compatible with vegan and other R packages, which means that they can be used in permutational tests (ie. PERMANOVA and PERMDISP) and in ordinations (ie. NMDS).

A similar type of partitioning can be used with abundance data, such as data that has not been relativized to account for total abundance at each site. The function for this partitioning is called ‘beta.multi.abund’ and it works the same way as the function shown above. Here, the function computes total abundance-based dissimilarity and component dissimilarities due to balanced variation in composition and due to abundance gradients. The function ‘beta.pair.abund’ calculates the pairwise matrices for the partitioning, similar to above. Note that if the data are relativized beforehand, then dissimilarity due to the abundance gradient is zero.

```
multi.dune.bray = beta.multi.abund(dune, index.family="bray")
multi.dune.bray

pairs.dune.bray = beta.pair.abund(dune, index.family="bray")
dune.bray.bal = pairs.dune.pa$beta.bray.bal
dune.bray.gra = pairs.dune.pa$beta.bray.gra
dune.bray.tot = pairs.dune.pa$beta.bray
```