**Discriminant Function Analysis via CART**

Discriminant Function Analysis (also called Discriminant Analysis as well as a confusing array of other names [canonical variates analysis, multiple discriminant analysis, etc.]) is similar to Cluster Analysis. The two primary objectives of DFA are to (1) describe differences among groups that you’ve already identified (e.g. via Cluster Analysis) and (2) predict the likelihood that an entity of unknown origin can be correctly assigned to one of those groups based on a suite of discriminating characteristics. It is an eigenanalysis technique that maximally separates a fixed number of *a priori*-defined groups. These pre-defined groups is what separates DFA from other forms of analysis. Like the methods we’ve covered over the past few weeks, DFA is a form of ordination, where the objective is to find discriminant functions (linear combinations of variables that maximize the grouping of objects into separate classes) that best differentiate groups in your data. In DFA, the independent variables in your dataset are used as predictors of group membership (making the groups the dependent variables). DFA generates synthetic variables (the discriminant functions) that maximize the formation of well-separated clusters of objects. An eigenvalue associated with each discriminant function indicates its ability to separate objects (larger is better).

(There are several forms of DFA, including linear, quadratic, Gaussian, and others. Today we are focusing on linear discriminant analysis, which uses linear combinations of predictors to predict the class of a given observation. Linear discrim. analysis is recommended if your dataset is small, although how small is not specified.)

To determine whether our pre-defined groupings really differ from each other, and to identify which variables best account for those differences, we will use a decision tree method, employing R’s version of Classification and Regression Trees (CART).

In the previous lesson, we classified the Bryce Canyon data into discrete clusters based on species composition using a variety of classification algorithms. The objective of these classification efforts was to produce groups that were distinct. We evaluated the success of these efforts using a variety of measures, including within-cluster/among-cluster similarity ratios and mean silhouette widths. In this lesson we will use those clusters as our *a priori*-defined groups to determine the degree to which site membership in a cluster is predictable from environmental variables not used in the classification. Given the environmental variables of each group, can we predict community composition? Here we'll be predicting total community composition within groups.

**Assumptions:**

* Homogeneous within-group variances: this is often violated when sample sizes are unequal and small.
* Multivariate normality within groups: DFA is relatively robust to violations of this assumption, particularly with respect to skew, but it is sensitive to outliers.
* Linear relationships among variables: this is commonly violated with ecological community data.
* Prior probabilities: these are the sampling probabilities for each of your pre-defined groups. They are seldom known and so they are typically assumed to be proportional to the number of samples in each group.

**Steps:**

1. Typically, you first perform a Cluster Analysis to identify groups in your data.
2. You then perform a DFA, which creates discriminant functions (synthetic variables) that will maximize the separation of objects among different groups. These algorithms also allow for predictive classification of new objects of unknown provenance into one of the groups based on the values of the measured explanatory variables. The DFA is examined to determine whether objects drawn iteratively from the dataset can be correctly assigned to their appropriate group.
3. Next you need to check for misclassification rates. We will use a decision tree approach to do so, but there are many other methods out there.
   1. If the misclassification rates are low, then you can describe the groups based on contributions of individual variables, like we did in other forms of ordination.
   2. If the misclassification rate is high, then you may need to refine your groups, e.g. via a different Cluster Analysis algorithm, and then repeat from step 2 until the misclassification rate is sufficiently low (at least better than random).
   3. Also build a confusion matrix to determine which clusters are consistently confused with which other clusters.
4. One you have a misclassification rate that is sufficiently low, then you may wish to examine group means, standard deviations, or other statistics via standard univariate stats tests on individual variables.

We will do steps 1-3 today (and for your assignment). (Because step 4 is a straightforward extension, I am skipping it for brevity.) I present an example below to illustrate the process. It is an extension of Cluster Analysis. The function lda() in *MASS* can also be used to conduct a DFA, but the approach I present below is more intuitive and straightforward in my mind.

**Example:**

Open a new RStudio session (with your class working directory) with the following libraries:

*labdsv*

*MASS*

*MVA*

*optpart*

*stats*

*tree* – this is a new library, so install it first

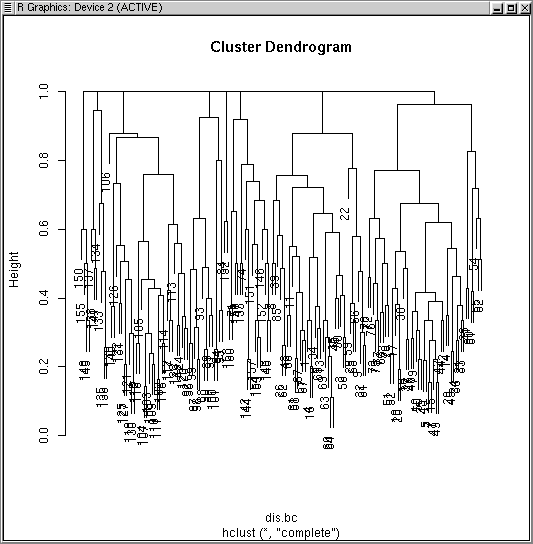
*vegan*

Load the Canyon site vegetation dataset (bryceveg.R) as an object named veg (with header = TRUE). Then use vegdist() to create a dissimilarity matrix named dis.bc, using the Bray-Curtis metric. (We are using the Bray-Curtis metric in this example because we have already used it in other lessons. You can play around with other metrics. For example, the Mahalanobis metric is often recommended in DFA because it is a metric that inherently tries to find separations. However, when I did a DFA with clusters of a Mahalanobis object, it was a mess! There were far too many groups with single members, no matter how I sliced the dendrogram, for CART to work with.)

First, we need to have our pre-defined groups, created via Cluster Analysis. I will create two different cluster outcomes. The first approach will be a hierarchical cluster analysis using the complete linkage or farthest-neighbor algorithm:

hcl1 <- hclust(dis.bc,method="complete")

plot(hcl1)



I'll slice the dendrogram at height = 0.99 to generate clusters:

hcl1.99 <- cutree(hcl1,h=0.99)

table(hcl1.99)

1 2 3 4 5 6 7 8

38 46 17 2 14 32 7 4

The result is eight clusters, with three of them (4, 7, and 8) being rather small.

The other approach is to use *optpart* (“optional partitioning”) to break the same dataset into 5 clusters. optpart() uses a single-linkage (nearest-neighbor) approach, the opposite of what we just did:

opt5 <- optpart(5,dis.bc)

table(opt5$clustering)

1 2 3 4 5

38 92 14 12 4

The result is five clusters (as requested), although one (cluster 5) is rather small. This tree (opt5) is a little simpler than hcl.99 since it was built around 5 clusters instead of 8.

So now that we have groupings (clusters), we can perform DFA, and use it to examine how the groups separate with respect to environmental variables. There are many ways of doing so (e.g. linear discriminant analysis, quadratic discriminant analysis, and others; check out candisc() in the *candisc* package for more info); today we will use a **decision tree approach** (a.k.a. **tree classifiers** method). Tree classifiers are often called **CART (Classification And Regression Trees)**, but technically, CART is a specific (copyrighted and trade-marked) example of such an approach.

Tree classifiers act like dichotomous keys (“If object has this property, it goes here; if not, it goes there”).

Install the package *tree*; in it, the tree classifier is called tree().

**Tree classifiers:**

Tree classifiers can be used to complement DFA (as we will do), but they can also be stand-alone analyses.

Load the brycesite.R site x environment dataset (with header=TRUE) as an object named site, and then attach(site).

To use the tree() function we use a formula with a dependent variable to the left of the ~ and independent (predictor) variables on the right:

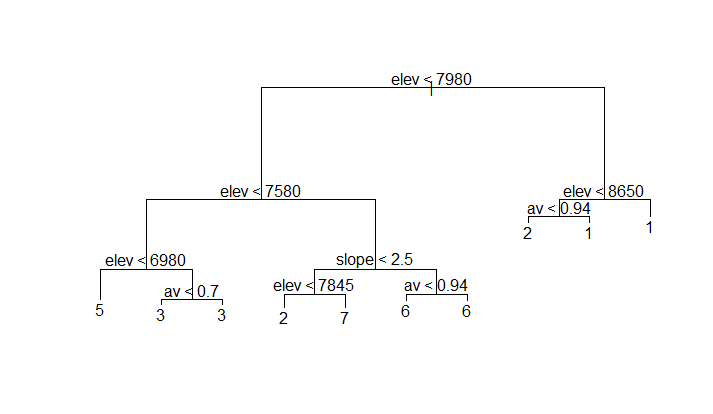
tree1 <- tree(factor(hcl1.99) ~ elev + av + slope + depth + pos)

The factor() argument is required so that the tree() function will recognize hcl1.99 as a categorical variable rather than numeric. (Don’t worry about the warning message here.)

The tree can be plotted and labeled with the following sequence:

plot(tree1)

text(tree1)



The tree grows downward: the “root” at the top, and each sequential split along each branch as you travel down the tree is labeled with the respective criterion for splitting. Values that are true go left; values that are false go right. The height of the vertical bar above each split reflects the decrease in **deviance** associated with that split. Deviance is similar to variance in that it assesses the variability around an estimate or model; as such, it can be used as an assay of goodness of fit, with low values of deviance being the goal. We’ve seen before (in the lesson on CCA) that adding variables can improve a model, but with a danger of overfitting. The addition of each variable should reduce the deviance, but at the price of a degree of freedom.

To see the output, just type its name:

tree1

node), split, n, deviance, yval, (yprob)

\* denotes terminal node

1) root 145 501.900 2 ( 0.25517 0.27586 0.08966 0.01379 0.08966 ... 0.01379 )

2) elev < 7980 83 273.200 6 ( 0.00000 0.18072 0.15663 0.02410 ... 0.02410 )

4) elev < 7580 34 87.550 3 ( 0.00000 0.14706 0.38235 0.02941 ... 0.05882 )

8) elev < 6980 14 7.205 5 ( 0.00000 0.00000 0.07143 0.00000 ... 0.00000 ) \*

9) elev > 6980 20 41.320 3 ( 0.00000 0.25000 0.60000 0.05000 ... 0.10000 )

18) av < 0.7 15 24.100 3 ( 0.00000 0.26667 0.66667 0.06667 ... 0.00000 ) \*

19) av > 0.7 5 10.550 3 ( 0.00000 0.20000 0.40000 0.00000 ... 0.40000 ) \*

5) elev > 7580 49 95.200 6 ( 0.00000 0.20408 0.00000 0.02041 ... 0.00000 )

10) slope < 2.5 20 48.230 2 ( 0.00000 0.40000 0.00000 0.05000 ... 0.00000 )

20) elev < 7845 10 12.220 2 ( 0.00000 0.70000 0.00000 0.00000 ... 0.00000 ) \*

21) elev > 7845 10 18.810 7 ( 0.00000 0.10000 0.00000 0.10000 ... 0.00000 ) \*

11) slope > 2.5 29 14.560 6 ( 0.00000 0.06897 0.00000 0.00000 ... 0.00000 )

22) av < 0.94 24 0.000 6 ( 0.00000 0.00000 0.00000 0.00000 ... 0.00000 ) \*

23) av > 0.94 5 6.730 6 ( 0.00000 0.40000 0.00000 0.00000 ... 0.00000 ) \*

3) elev > 7980 62 83.610 1 ( 0.59677 0.40323 0.00000 0.00000 0.00000 ... )

6) elev < 8650 45 61.830 2 ( 0.44444 0.55556 0.00000 0.00000 ... 0.00000 )

12) av < 0.94 34 42.810 2 ( 0.32353 0.67647 0.00000 0.00000 ... 0.00000 ) \*

13) av > 0.94 11 10.430 1 ( 0.81818 0.18182 0.00000 0.00000 ... 0.00000 ) \*

7) elev > 8650 17 0.000 1 ( 1.00000 0.00000 0.00000 0.00000 ... 0.00000 ) \*

This output takes a little explaining (and a little manipulating to fit; the ellipses […] represent class probabilities I've deleted to make this all fit on a page in portrait orientation). The tree is a dichotomous key (similar to a dichotomous taxonomic classification key). The first line is the root node (which you can think of as the trunk of the tree), which indicates that:

* there are 145 plots (plots with missing [NA] values were dropped out, so it's less than the 160 sites in the dataset)
* the null deviance is 501.900
* the null type of cluster (most likely) is 2
* the respective probabilities for the 8 clusters are 0.22517, 0.27586, ... etc.

From that root line in your output, you can see that the maximum probability is for cluster 2 = 0.27586.

Had we followed branch 3 (which I’ve highlighted in yellow to make it easier to see),

* there are 62 plots
* the deviance is 83.610
* the most likely cluster is 1

Note that the sum of the respective deviances at the first split after the root (denoted by the tabbed entries, i.e., 2 and 3) is less than the null deviance, i.e., 273.200 + 83.610 < 501.900. In fact, we achieved a reduction in deviance of 501.90 - (273.200 + 83.610) = 145.09 with just a single split. The objective of the tree classifier is to minimize the deviance of the tree with each split, and the optimal split is calculated by comparing reductions in deviance. No other possible split in these data would have achieved as significant a reduction in deviance as achieved here.

We now simply proceed the same for each remaining branchings (splits). For example, on branch 2, where we now have 83 plots, the next split is

4) elev < 7580 34 87.550 3 ( 0.00000 0.14706 0.38235 ... 0.00000 0.05882)

5) elev > 7580 49 95.200 6 ( 0.00000 0.20408 0.00000 ... 0.14286 0.00000)

Branches continue splitting until (i) they become purely homogeneous or (ii) they become too small to split. In tree(), the minimum "terminal node" size is 10, and the minimal number of plots to be split off is 5. Once the tree is finished splitting, terminal nodes are designated with an asterisk at the end of the probability vector, as seen for nodes 8, 18, and 19 (among others) in the tree above. Terminal nodes are equivalent to coming to the identification of a species in a taxonomic key, and predict the type for all sample in that node.

To see a simple summary of the results, enter:

summary(tree1)

Classification tree:

tree(formula = factor(hcl1.99) ~ elev + av + slope + depth + pos)

Variables actually used in tree construction:

[1] "elev" "slope" "av"

Number of terminal nodes: 10

Residual mean deviance: 0.9841 = 132.8 / 135

Misclassification error rate: 0.2069 = 30 / 145

Several important pieces of information are given:

* Although there were five variables in the model, only elev, slope, and av were used (depth and pos were not important; variables that are never used in a split are dropped).
* Although there were only 8 clusters to predict, there are 10 terminal nodes.
* The misclassification error rate is 20.69%, which is 30 out of 145.

Let’s cover each of these in more detail now. First, if we gave the classifier five variables to use, why did it only use three? At each split, the best variable (that gives the maximum reduction in deviance) is used, and a single variable can be used more than once. In this example, soil depth and topographic position were never used and so were dropped from the analysis. In contrast, elev was used repeatedly.

If there were eight clusters, why are there 10 terminal nodes? For some types there is more than one route to a terminal node. For example, nodes 22 and 23 (among others) both predict cluster 6, but nodes 22 and 23 are a single split, and yet both predict cluster 6. Why is that? Node 22 predicts type 6 perfectly (p = 1.0, dev. = 0) whereas node 23 is less reliable (p = 0.60, dev. = 6.73). Splitting node 11 into 22 and 23 achieved a reduction in deviance of 14.560 - (6.730 + 0.000) = 7.83 with a single split, even though both branches predicted the same type. Less obvious at first glance is that some clusters were never predicted at all: None of the terminal nodes predicted clusters 4 or 8. There were too few plots in those types to work with, and they fell below tree()’s minimum splitting rule of at least 5 plots/type.

The **misclassification error rate** was 30 out of 145 = 0.2069 or 20.69%. That's not great (basically 1 out of 5 entries would be misclassified). Surprisingly, there is no consensus as to what a good error rate is. Certainly you would want something better than random (i.e., < 50%). But in addition to the overall misclassification error rate, there is also the matter of determining which types were confused with which other types. We can view this in a table called a **confusion matrix**. To create such a matrix, we need the vector of predicted type for each plot. This we obtain as follows:

tree1.pred <- predict(tree1,newdata=site,type="class")

tree1.pred

[1] 1 1 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2 3 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2

[34] 2 1 2 3 2 1 2 2 2 2 2 2 1 2 1 2 2 6 7 3 2 2 1 2 1 1 1 1 1 1 2 1 1

[67] 1 2 1 3 3 3 3 3 8 3 3 2 2 6 6 2 2 3 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5

[100] 5 5 6 2 6 2 6 6 2 2 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 7

[133] 7 7 7 7 7 7 6 6 7 3 3 3 3 3 3 3 3 8 5 8 3 3 3 3 5 5 5 3

Levels: 1 2 3 4 5 6 7 8

(Again, ignore the warning.) The predict function takes an existing tree and "keys out" the predicted membership of a set of plots (sites). It's often used to predict the membership of plots that weren't used in the model (as suggested by the name newdata= but can be used with the original data to get a vector of predicted types. Then we can use the table() function to create the confusion matrix:

table(tree1.pred,hcl1.99)

hcl1.99

tree1.pred 1 2 3 4 5 6 7 8

1 26 2 0 0 0 0 0 0

2 12 31 0 0 0 4 0 0

3 0 8 12 1 0 0 0 3

4 0 0 0 0 0 0 0 0

5 0 0 4 0 14 0 0 0

6 0 3 0 0 0 27 0 0

7 0 1 0 1 0 1 7 0

8 0 1 1 0 0 0 0 1

The actual values are in the columns and the predicted values are in the rows. The diagonals are thus the correct assignments. You can see some confusion between types 1 and 2 (2 + 12 = 14 errors) and 2 and 6 (4 + 3 = 7 errors), for example, and you can see where the missing types (4 and 8) are allocated (4 was misclassified into clusters 3 and 7, for example). (It’s called a confusion matrix because you can think of it as cluster 1 was confused with cluster 2, etc.)

Recall from the lesson on Cluster Analysis that the complete linkage (farthest-neighbor) algorithm as used above overestimates differences among groups whereas the single linkage (nearest-neighbor) approach underestimates them. You will use the single linkage approach in your homework assignment and compare its output to that generated above.

**Final notes on DFA:** The true discriminatory power of your assay will be found only when it is tested with a completely independent sample. If you have a very large dataset, it is best to hold back some of your original data (the normal rule of thumb is a random 10%) and then use that to check and see how well your model performs. This is called **cross-validation**: You use 90% of your data to train your model and the withheld 10% to test it. That way the test is independent because those data weren’t used to construct the model.

The examples we worked through were to examine overall communities by sites, but DFA can also be used to separate any groupings. For example, suppose you collect an individual that you suspect may belong to a new species; it’s similar to known species but is just different enough to make you wonder. You can use DFA on morphometric traits (from your individual compared to individuals of known species) to determine how distinct your individual is, and how likely it is to be misclassified as a known species. That kind of analysis is better performed with lda() in *MASS*.

**Assignment:** due 0800 Monday, May 3

Today, it will be easier if you do not start a fresh RStudio session because you will be extending work that you just did.

**Q1. Compare what we did in the first example above to a classification created via the single-linkage method, which resulted in a 5-cluster object we called opt5. Was the misclassification rate higher or lower than the complete/farthest-neighbor method? Which classes were most misclassified?**

**Q2. Now perform a DFA on the grassland.community.csv data we performed cluster analysis on last time (using the hierarchical average linkage method), with the four numerical variables from plot.metadata.csv.** Some important notes:

-Do not attach the plot.metadata.csv object.

-When you build your tree classifier with a hierarchical object rather than optpart, you have to specify $labels in

yourtreename <- tree(factor(yourRobject$labels) ~ etc.

-Because there are 27 sites in the grassland community data, the confusion matrix will be complicated, so you can skip it.

**How many clusters were formed? Of the four variables, which ones were used? What was the misclassification rate?**

Make an RMarkdown Word file of your work and turn that in. Be sure to include your answers to the questions asked! Turn in your assignment as a Word document via email to [iroro.tanshi@ttu.edu](mailto:iroro.tanshi@ttu.edu) no later than 8:00 a.m. on Monday of next week. In your email, please include the following as the Subject line:

Assignment on DFA

**Final course thoughts:**

This class covered some (but by no means all!) multivariate statistical approaches to analyzing data from ecological communities, using the R freeware statistical analysis package. R is perhaps the most widely used stats analysis package now, having replaced SAS, JMP, SYSTAT, SPSS, etc. in all government and non-profit agencies because of its price (free), transparency, and flexibility. Thus, this course has been designed as a kind of “buy one, get one free” class: learn stats, and learn some R alongside.

There are MANY other R packages out there that can do many of the analyses we did, with various other options, and there are LOTS of other ways of generating more attractive and informative plots. I encourage you to explore what packages might be useful to you in analyzing your own data.

And there are MANY other kinds of analyses that are relevant to understanding ecological communities. We covered the foundational analyses, but in no great depth. Your own research objectives may require analyses that we did not cover (such as various forms of ordination).

So even though we’re at the end of this course, you all are really at the start of your statistical analyses of ecological communities.