Multivariate Regression Trees

June-Hung Chen
Dr. Paul Speckman, Project Advisor

Department of Statistics
University of Missouri, Columbia
December 9, 1998
Acknowledgements

I would like to thank Dr. Paul Speckman for supervising this project and serving as chairman of the project committee. His guidance, support, and patient were crucial to the completion of this paper and were greatly appreciated.

I would also like to thank Dr. David Larsen and Jennifer Grabner from the School of Natural Resources, who offered the data set from the study of vegetative species abundance, and Dr. Tony Sun for taking the time to serve on my examination committee.
The regression tree is an exploratory technique for uncovering the structure of data, especially when the relationship between response variables and explanatory variables is complex. The regression tree is fit by binary recursive partitioning whereby a data set is successively split into increasing homogeneous subsets until it is infeasible to continue. Each node of the tree consists of an inequality condition on one of the independent variables. In this paper, we extend this algorithm to multivariate response variables. An important aspect at the multivariate regression tree is that it partitions a data set into nodes that are relatively homogeneous with respect to several dependent variables at once. This partition can be used to classify the data set. An example is given for a vegetative species abundance study. An S program has been written to implement multivariate trees.
Contents

1. Introduction 1

2. Literature Review 3
   2.1 Background 4
   2.2 Algorithm 9
   2.3 Related works

3. Multivariate Regression Trees 13
   3.1 Method
   3.2 Splus implementation

4. Application to Problem in Forestry 17
   4.1 Data description 19
   4.2 Results

References 23

Appendix I. Figures

Appendix II. Splus Programs for Multivariate Regression Trees

Appendix III. Splus Programs for Application to Problem in Forestry
1. Introduction

In recent years, tree-based methods for regression and classification have become powerful tools of data analysis. The regression tree is an exploratory technique for uncovering the structure of data. It is useful when there are a large number of explanatory variables and a complex relationship between the response variable and the explanatory variable is expected. This model also provides an alternative to linear and additive models for regression problems and to linear logistic and additive logistic models for classification problems. The trees are fitted by binary recursive partitioning whereby a data set is successively split into increasingly homogeneous subsets until it is infeasible to continue. Each node of the tree consists of an inequality condition on one of the independent variables. The partitions induced by the tree structure may be interpretable by the researcher in their own right as groups understood to have a commonality of behavior. It also provides a way to encapsulate and structure the knowledge of experts to be used by less-experienced users.

According to Clark and Pregibon (1992),

This method is gaining widespread popularity as a means of devising prediction rules for rapid and repeated evaluation, as a screening method for variables, as a diagnostic technique to assess the adequacy of linear models, and simply for summarizing large multivariate datasets. Some possible reasons for its recent popularity are that:

- In certain applications, especially where the set of predictors contains a mix of numeric variables and factors, regression trees are sometimes easier to interpret and discuss than linear models;
• Regression trees are invariant to monotone reexpressions of predictor variables so that the precise form in which these appear in a model formula is irrelevant;
• The treatment of missing values (NAs) is more satisfactory for regression trees than for linear models; and
• Regression trees are more adept at capturing nonadditive behavior; the standard linear model does not allow interactions between variables unless they are prespecified and of a particular multiplicative form.
2. Literature Review

2.1 Background

The construction of decision trees dates from work in the social sciences by Morgan and Sonquist (1963). An implementation of their ideas was realized in the computer program AID (Automatic Interaction Detection), which served to stimulate much subsequent research, such as THAID (Morgan and Messenger, 1973) and CHAID (Kass, 1980). The primary difference among these methods is the stopping rule for the growing of trees. In the statistics community, Breiman, Friedman, Olshen, and Stone (1984) had a seminal influence both in bringing the work to the attention of statisticians and in proposing new algorithms for constructing trees. They called their algorithm CART, for Classification and Regression Trees. Their primary innovation was not to limit node expansion in the tree-growing process. They recommended growing an overly large tree and then pruning some nodes off. Subsequent work by Chou, Lookabough, and Gray (1989) generalized this concept to other tree functions besides tree size. Their other important innovation was the introduction of surrogate splits to provide a mechanism to grow trees and make predictions in the presence of missing values and also to provide a measure of variable importance. The instruction within S of tree-based models described by Clark and Pregibon (1992) has made the methods much more available. Their methods are very much in the spirit of exploratory data analysis, with many functions to investigate trees.
Our methodology parallels that of Ciampi, Chang, Hogg, and McKinney (1987) in the use of the deviance function as the basis for choosing partitions and also parallels that of Breiman et al. in tree growing, pruning and subsequent cross-validation procedures.

2.2 Algorithm
The CART algorithm consists of a base algorithm that is applied recursively. The base algorithm generates a tree with a single node and two leaves. The node divides the original data set into two mutually exclusive data sets. The base algorithm can then be applied to each of these two data sets, generating a tree with four leaves. In this manner, a tree of any depth can be generated.

A. Growing the tree
Suppose we have n observations, p different predictors $X_1, X_2, ..., X_p$ and a single-response variable $Y$. For regression, the "deviance" of a node is usually defined as

$$D_{\text{node}} = \sum_{\text{node}} (y_i - \bar{y})^2,$$

which is the total sum of squares for all observations in the node. The total deviance of a tree is

$$D(T) = \sum_{\text{terminal nodes}} D_{\text{node}}.$$

Define the LS (least squares) split function for node $g$ as

$$\Delta_s(T) = D_g - D_{gL} - D_{gR},$$
where $g_L$ and $g_R$ are “left” and “right” subnodes of $g$. The deviance for the tree $T_g$ consisting of $g$ and the two subnodes $g_L$ and $g_R$ is

$$D(T_g) = D_{g_L} + D_{g_R},$$

which is the sum of the deviances of the left and right leaves. Then for each node, we do the following steps recursively until every node is a terminal node:

1. Define $D_0 = \sum (y_i - \bar{y})^2$, the deviance of the root node. If deviance < “mindev” (nominally set to $0.01 \times D_0$) or node size < $t$, then we stop splitting this node. Here, $t$ is a prespecified parameter, which we set nominally to 10.

For each continuous independent variable $X_j$:

(i) Sort the $X_j$'s in the node into unique, ascending values $Z_1, \ldots, Z_n$ ($\ell \leq n$).

Consider candidate cut points for the node given by $c_k = \frac{Z_k + Z_{k+1}}{2}$, where $k = 1, \ldots, \ell-1$.

(ii) For $k = 1, \ldots, \ell-1$, divide the data set into two nodes, $X_{ij} < c_k$ and $X_{ij} > c_k$. If either node has less than $t/2$ observations, we do not attempt to split at $c_k$. Otherwise, compute total deviance for the node:

$$D(T_g) = D_{g_L} + D_{g_R},$$

where $g_L = \{i \in g : X_{ij} < c_k\}, g_R = \{i \in g : X_{ij} > c_k\}$.

(iii) The candidate split point for variable $X_j$ is the $c_k$ which minimizes $D(T_g)$.

For each categorical independent variable $X_j$:
(i) Find the unique values $Z_1, \ldots, Z_l$ ($l \leq n$) of the levels of $X_j$ in the node.

(ii) Consider each of the distinct $2^{l-1} - 1$ partitions of $Z_l$'s into two mutually exclusive groups. If either node has size less than $t/2$, we do not consider this split. Otherwise, we calculate

$$D(T_g) = D_{L} + D_{R}.$$  

(iii) The candidate split is the partition that minimizes $D(T_g)$.

2. If possible, split $g$ into two nodes, $g_L$ and $g_R$, using the variable and split which minimize $D(T_g)$ over all continuous and categorical $X_j$'s.

B. Efficient algorithms

There are two approaches that improve the computing efficiency for this split function.

1. Update algorithm for continuous variables (see Breiman et al., 1984).

Assume the $X_j$ are ordered and let $D_i$ denotes the candidate deviance $D_{LL} + D_{RR}$ when $g_L$ is the first $i$ data points. Then $D_{i+1}$ can be computed in one step with the update formula

$$D_{i+1} - D_i = \left( \frac{i}{i+1} \right) \left( \bar{y}_{i+1} - \bar{y}_L \right)^2 - \left( \frac{n-i-1}{n-1} \right) \left( \bar{y}_{i+1} - \bar{y}_R \right)^2,$$

where $\bar{y}_L = \frac{1}{i} \sum_{j=1}^{i} y_j$ and $\bar{y}_R = \frac{1}{n-i} \sum_{j=i+1}^{n} y_j$. In this way, deviances for all the candidate splits for a node of size $n$ can be computed in $O(n)$ operations.
2. For categorical variables, it can be shown that we only need to consider $\ell - 1$ partitions (see Breiman et al., 1984). Assume that the $Z_k$'s have been ordered so that if

$$y_k = \frac{\text{mean}(y_i)}{x_i = z_k},$$

then $\bar{y}_1 \leq \bar{y}_2 \leq \ldots \leq \bar{y}_\ell$. It then suffices to consider splits of the form $g_L = \{ i \in g : X_i \in \{Z_1, \ldots, Z_k\} \}$ and $g_R = \{ i \in g : X_i \in \{Z_1, \ldots, Z_k\} \}$, $k = 1, \ldots, \ell - 1$. This reduces the search for the best subset of categories from $2^{\ell - 1} - 1$ to $\ell - 1$ categories.

C. Pruning the tree

Since tree size is intentionally not limited in the growing process, we may overfit the data. One approach is to prune the tree to a suitable size with a procedure very similar to backward variable selection in general regression models. The methodology of tree pruning was first introduced by Breiman et al. (1984). They noted that for any split of $g$ into $g_L, g_R$, $D_g \geq D_{g_L} + D_{g_R}$. Thus, the more splitting done, the better total deviance looks, even if the added split only overfits the data. Define the cost-complexity measure

$$D_{\alpha}(T') = D(T') + \alpha \text{size}(T'),$$

where $D(T')$ is the deviance of the subtree $T'$, $\text{size}(T')$ is the number of terminal nodes of $T'$, and $\alpha$ is a cost-complexity parameter. For any specified $\alpha > 0$, cost-complexity pruning determines the subtree $T'$ that minimizes $D_{\alpha}(T')$ over all subtrees of $T$. As with model fitting, the deviance decreases as tree size increases. Thus, we can get a better tree by looking for a big change in deviance at a particular
value of $\alpha$. If $\alpha = 2\alpha^1$, the cost-complexity measure $D_\alpha(T)$ is equivalent to AIC for regression.

Another method is to use a new data set to estimate the risk of tree structured procedures. In this case, the subtree that minimizes the sum squared prediction error determines the best tree. However, this approach requires that one set of sample data be used to construct the procedure and a disjoint set be used to evaluate it. When the combined set of available data contains a thousand or more cases, this is a reasonable approach. But if only a few hundred cases or less in total are available, it can be inefficient in its use of the available data. Thus, the another kind of cross-validation is preferable.

D. Cross-validation

The basic idea of cross-validation is to randomly divide the data set into several mutually exclusive and roughly equally subsets. For each subset, a subtree $T'$ is grown by fitting the remaining subsets. Then $T'$ is used to predict the subset that has been held out. For instance, suppose we split the training set into 10 equally sized parts. We can then use 9 of them to grow the tree and test it on the tenth. This can be done in 10 ways, and we can average the results. The process is repeated for all subsets, and the deviances for the best subtrees of each size from each set are accumulated. Define the cross-validation error for a subset $V_i$ that has been held out of subtree $T'$ as
\[ CV_i(T') = \frac{\sum \{ y_j - \hat{y}_j \}^2}{n_i}, \quad i = 1, \ldots, m, \]

where \( n_i \) is the size of \( V_i \), \( m \) is the number of subsets, and \( \hat{y}_j \) is the predict value for \( y_j \).

Define the total cross-validation score as

\[ CV(T') = \sum_i CV_i(T'). \]

The subtree of the size that minimizes \( CV(T') \), which is equivalent to minimizing the total deviance, can be treated as the best tree for the data set. More generally, allowance can be made for the ultimate desire for simplicity as well as accuracy.

### 2.3 Related work

#### A. Optimal Partitions for Regression Trees

Since the CART algorithm exhaustively searches all possible cut points for each independent variable to process the growing of the tree in each node, it can take a long time to find the optimal partition. This is especially true if a deviance other than squared error less is used and the efficient steps from Section 2.2 are not available.

Since the number of possible partitions grows exponentially in \( \ell \), this approach is impractical when \( \ell \) is larger than about 10 or 20 for general deviance measures.

In 1991, Chou presented an iterative algorithm that finds a locally optimal partition for an arbitrary loss function, in time linear in \( \ell \) for each iteration. The algorithm is a K-means like clustering algorithm that uses as its distance measure a generalization of Kullback’s information divergence. Moreover, he proved that the
globally optimal partition must satisfy a nearest neighbor condition using divergence as the distance measure. These results generalize similar results of Breiman et al. to an arbitrary number of classes or regression variables and to an arbitrary number of bins. He also suggested some additional applications of the algorithm, including the design of variable combinations, surrogate splits, composite nodes, and decision graphs.

B. Tree-Structured Method for Longitudinal Data

In 1992, Segal extended the regression tree model to the multiple response setting and to longitudinal data in particular. He used the regression tree to identify strata with common covariate values and homogeneous multiple outcomes. He assumed that each individual observation has a $T \times 1$ vector of responses $y_i = (y_{i1}, y_{i2}, ..., y_{iT})$ and defined $V(\theta, g)$ to be the model covariance matrix of the response for node $g$ depending on an unknown parameter $\theta$. By replacing the deviance with

$$SS(g) = \sum_{i \in g} (y_i - \mu(g))^t V(\theta, g)^{-1} (y_i - \mu(g)),$$

he introduced a new split function $\Delta(s, g) = SS(g) - SS(g_L) - SS(g_R)$ for any split $s$ of each possible $g, g_L$, and $g_R$. However, $\Delta(s, g)$ could be negative for different covariance matrices. By imposing $V(\theta, g) = V(\theta_L, g_L) = V(\theta_R, g_R)$, we can ensure $\Delta(s, g) \geq 0$ and produce a tree by maximizing $\Delta(s, g)$. Segal also proposed Hotelling's $T^2$ to test the split function that optimizes between node separation rather than within node homogeneity,

$$T^2 = \frac{N_L N_R}{(N_L + N_R)} (\mu(g_L) - \mu(g_R))^t S^{-1} (\mu(g_L) - \mu(g_R)),$$
where $N_L$ and $N_R$ are the sample sizes for the left and right children nodes, respectively, and $S$ is the pooled sample covariance matrix. Since heterogeneity in longitudinal data also can affect covariances, the split functions are applied to residuals after accounting for the mean structure.

C. Piecewise-Polynomial Regression Tree

In 1994, Chaudhuri, Huang, Loh, and Yao introduced a new nonparametric method of tree-structured regression called SUPPORT ("Smoothed and Unsmoothed Piecewise-Polynomial Regression Trees") and compare its prediction mean square error (PMSE) with CART and some spline-based methods like Linear Regression, Quadratic Regression, MARS etc. The estimate is typically made up of several pieces, each piece being obtained by fitting a polynomial regression to the observations in a subregion of the data space. By employing this piecewise-polynomial regression tree method, fewer splits are necessary than with the original piecewise linear regression tree. This can aid interpretation. Partitioning is carried out recursively as in the regression tree method. Chaudhuri et al. note that when deciding whether a node should be split, it is not enough to determine the change in PMSE after just one split. For instance, if the distribution of the data set is like a sine-wave with many cycles, it may take several splits of a node before rapid decreases are realized. This means we should use some techniques to deal with this sort of problem in advance. Their algorithm employs "local" cross-validation at each node. By dividing the data in the node into $K$ cross-validation samples, for each combination of $(K - 1)$ parts, a nested sequence of trees is
constructed and the remaining parts are used to determine if any of the trees in the sequence report a fractional reduction of PMSE greater than a user-specified fraction $f$ . If the proportion of times out of $K$ when the latter event occurs exceeds a pre-assigned threshold level $\eta$, the node $g$ is split. Split selection is done by computing the residuals from a linear model fitted to the node and comparing the distributions of the two subgroups associated with the positive and negative residuals along each independent variable. If the fit is adequate, the distribution should not be very different. The comparison is made by testing the sample means and sample variance differences, with the variable giving the most significant test statistic chosen to split the node. The cut-point is the average of the two sample means. And since the true regression function is known to be smooth, the polynomial pieces may be glued together by means of weighted averaging.
3. Multivariate Regression Trees

3.1 Method

The regression tree is a good approach for predicting a dependent variable using the information from several independent variables. But in reality, we may consider several dependent variables simultaneously. We can extend the simple regression tree to a multivariate regression tree. In this section, we use Segal's (1992) idea for other kinds of multivariate data, not necessarily longitudinal structured. Suppose we have dependent variables $Y_1, Y_2, \ldots, Y_d$ and independent variables $X_1, X_2, \ldots, X_p$, which can be continuous or categorical. Since the data are not longitudinal, we don't have a parametric covariance structure as in Segal (1992). We simplify our deviance defining the deviance for node $g$ with respect to a dependent $Y_k$ as

$$D_k(g) = \sum_{i'} (Y_{rk} - \bar{Y}_k)^2$$

and the total deviance for the node $g$ as

$$D(g) = \sum_k w_k D_k(g) = \sum_k \sum_{i'} w_k (Y_{rk} - \bar{Y}_k)^2,$$  \hspace{1cm} (3.1)

where $w_k$ is a scale factor. We typically take $w_k = 1/\text{Var}(Y_k)$. This definition of deviance is used to produce a regression tree based on the multivariate data set. The growing algorithm of the multivariate regression tree has the same structure as the algorithm in Section 2.2 but with deviance defined by (3.1). This leads to the following changes.

For a continuous independent variable $X_j$, update formula (2.1) is replaced by
\[ D_{i+1} - D_i = \sum_{k=1}^{d} w_k \left( \left( \frac{i}{i+1} \right) \bar{y}_{i+1,k} - \bar{y}_{L,k} \right) - \left( \frac{n-1}{n-i-1} \right) \left( \bar{y}_{i+1,k} - \bar{y}_{R,k} \right)^2, \]

where \( \bar{y}_{L,k} = \frac{1}{i} \sum_{t=1}^{i} y_{tk} \) and \( \bar{y}_{R,k} = \frac{1}{n-i} \sum_{t=i+1}^{n} y_{tk} \).

For a categorical independent variable \( X_j \), the efficient method of Section 2.2 unfortunately no longer applies because in general it is impossible to reorder the distinct levels \( Z_1, \ldots, Z_\ell \) of the \( X_j \)'s to have the means of each dependent variable ordered. Thus an exhaustive search must be performed over all \( 2^{\ell-1} - 1 \) partitions of \( Z_\ell \)'s. For each candidate split, we calculate the new deviance \( D(T_{k}) \) for the node as follows.

Define

\[ U_{kj} = \sum_{x_j = Z_k} y_{ij}, \quad j = 1, \ldots, d \text{ and } k = 1, \ldots, \ell. \]

Since the deviance in a node can be written as

\[ D(g) = \sum_{j} \sum_{i \in g_j} w_j (y_{ij} - \bar{y}_j)^2 = \sum_{j} \sum_{i \in g_j} w_j y_{ij}^2 - \sum_{j} \sum_{i \in g_j} w_j \left( \frac{\sum_{i \in g_j} y_{ij}^2}{n} \right)^2, \]

where \( n \) is the node size, the deviance after splitting into \( g_L \) and \( g_R \) is

\[ D(T_{k}) = D_{gL} + D_{gR} = \sum_{j} \sum_{i \in g_L} w_j y_{ij}^2 - \sum_{j} \sum_{i \in g_R} w_j \left( \frac{\sum_{k \in g_L} U_{kj}}{n_L} + \frac{\sum_{k \in g_R} U_{kj}}{n_R} \right), \]

where \( n_L \) and \( n_R \) are the sizes of subset \( g_L \) and \( g_R \), respectively. Define the reduction term.
\[ \Delta_{t_L, t_R} = \sum_j w_j \left( \frac{\left( \sum_{k \in t_L} U_{kj} \right)^2}{n_L} + \frac{\left( \sum_{k \in t_R} U_{kj} \right)^2}{n_R} \right) \]

The candidate split is the partition that maximizes the value of \( \Delta_{t_L, t_R} \).

The variable and split which minimizes \( D(T_q) \) over all continuous and categorical \( X_j \)'s is chosen. By recursively using this algorithm until every node is a terminal node, we obtain a multivariate regression tree. We can then use the pruning and cross-validation procedures, which are the same as for simple regression trees, to get an optimal tree.

### 3.2 Splus implementation

We have written an Splus program to implement the multivariate regression tree. In Splus, the `mult.tree` function generates an object of class “mtree”.

```splus
> mtree <- mult.tree(Y~x1+x2+...+xd, ywt, ...)
```

The first argument of the function is a model formula, with the `~` symbol standing for “is modeled as”, \( Y \) is the dependent variables matrix, \( x1,...,xd \) are independent variables, and \( ywt \) is a vector of dependent variable weights. After getting a multivariate regression tree, we can use the `prune.mlt.tree` function to prune the full tree `mtree` down to a suitable subtree “subtree1”. For example,

```splus
> subtree1 <- prune.mlt.tree(mtree, k=a, est=b, ...)
```
where $k$ is the value for cost-complexity parameter $\alpha$, and $\text{best}$ is the integer specifying the size (i.e. number of terminal nodes) of a specific subtree in the cost-complexity sequence to be returned. On the other hand, we can use the \texttt{cv.mult.tree} function to perform cross-validation procedures by

$$> \text{cv.subtreel} \leftarrow \text{cv.mult.tree}(\text{mtree}, \text{size}=c, \ldots)$$

where \texttt{size} is the number of terminal nodes in each tree in the defining sequence.
4. Application to Problem in Forestry

4.1. Data description

A. Background

We obtained this data set from Jennifer Grabner in the School of Natural Resources of the University of Missouri - Columbia. The data were collected from 275 plots during the summer of 1997. Plots were located in Shannon, Carter, and Oregon counties of the southeastern Missouri Ozarks. The aim of the survey was to identify relationships between the vegetation and the environment (specifically geology, landform, and soils) and to develop an ecological classification of the landscape based on patterns in geology, landform, soils, and the resulting vegetation. The data set is large and the relationship between response variables and independent variables is complex. Since multivariate regression tree is a very good approach to deal with this sort of data sets and it’s easy to interpret, we choose to use a multivariate regression tree to proceed the analysis.

B. Variable description

The variables in the data set are:

1. Environmental variables (independent variables, all categorical)

   phase = whether plot (10x20meter) was in unit with deep soil (dp) or soil with shallow to variable depth to bedrock (vd).

   lta = Landtype Association, layer in national hierarchical classification system - scale is about 1:100,000 map scale; based on local climate, topography, geology, soil groups, broad vegetation patterns
geo = geology (Roubidoux, Upper and Lower Gasconade, Eminence) - Ordovician and Cambrian aged materials of sandstone, chert, and dolomite

pos = relative hill slope position (upper, upper-middle, middle, lower-middle, lower)

If = landform, a unit of land defined by its position in landscape and its surface shapes (convex, concave, linear,...) (summit, shoulder, shoulder-ridge, bench, back slope, foot slope,...)

asp = aspect class, aspect = azimuth in degrees of compass at which slope of hill is facing the sun (pro (protected) = 340-70 deg on compass; nte (neutral-east) = 71-159 deg; exp (exposed) 160-250 deg.; ntw (neutral-west) = 251-339 deg.)

ord = soil taxonomic order (ultisol, alfisol, mollisol, inceptisol) (ultisol = highly weathered soils, low base-saturation = relatively low fertility; alfisol = > 35% base saturation = rel. higher fertility; mollisol = thick, dark surface horizon and high base saturation = high fertility)

2. Dependent variables (percent of plot covered by each of the following species)

vaccvaci = Vaccinium vacillans - lowbush blueberry - family Ericaceae; common shrub in dry, acidic sites (perennial)

desmglut = Desmodium glutinosum - fat-leaved tick trefoil - family Leguminosae - common legume in moist rich sites (perennial)

astepate = Aster patens - spreading aster - family Compositae - common dry woodland aster (perennial)

caredigi = Carex digitalis - sedge - family Cyperaceae - relatively common sedge in somewhat moist, protected sites (perennial)

desmrotu = Desmodium rotundifolium - round-leaved tick trefoil - family Leguminosae - somewhat common legume in acidic woodlands (perennial)

euphcoro = Euphorbia corollata - flowering spurge - family Euphorbiaceae - common Ozark woodland forb (perennial)

lespinte = Lespedeza intermedia - Lespedeza - family Leguminosae - common woodland legume (perennial)
monaruss = Monarda russeliana - horsemint - family Labiatae - forb often found in acidic, open woodlands (perennial)

panicomm = Panicum commutatum - panic grass - family Gramineae - grass commonly found on variable site conditions (perennial)

phrylept = Phyma leptostachya - lopseed - family Phrymaceae - forb is only species in this family - found in relatively moist rich sites (perennial)

smilbona = Smilax bona-nox - greenbriar - family Liliaceae - thorny woody vine is only found in relatively base-rich soils where dolomite bedrock is close to or near the surface (perennial)

smilrace = Smilacina racemosa - false Solomons’s seal - family Liliaceae - forb occurs in rich, protected sites (perennial)

The data for the 12 dependent variables are numerical but were all grouped in one of six categories with midpoints (0, 0.01, 0.5, 3, 10, 20). Since the difference between any two possible values within each dependent variable is quite big, we decided to try a transformation on all the dependent variables. We tried several transformations. The square root transformation with transformed levels (0, 0.1, 0.71, 1.73, 3.16, and 4.47) appeared to give satisfactory spread of the data. We applied this transformation to all 12 dependent variables and used this in the following analysis.

4.2 Results

First, the data set was fit using the mult.tree function. A full multivariate regression tree $T_1$ was grown by successfully splitting so as to minimize $D(T_1)$ based on the data set. In $T_1$, there are 41 nodes, and some of the nodes just contain only a few observations (less than 6). Thus, we tried to prune the full tree $T_1$. 

19
Because the cross-validation procedure is based on a random partition of the data set into 10 parts, different cross-validation runs produce different results. By using the `cv.mult.tree` function, we ran cross-validation three times. In the plots of cross-validation, we can see that these three plots indicate somewhere around 5 or 12 nodes are suitable choices for this data set since they appear to produce the smallest overall prediction error. We used the `prune.mult.tree` function to cut the full tree down to two subtrees $T_5$ and $T_{12}$, which have 5 nodes and 12 nodes respectively. Since some nodes of $T_{12}$ contain only a few observations (less than 8) and since $T_5$ may have less prediction error and is easier to interpret, we choose $T_5$ as the optimal tree to proceed with subsequent analyses.

The layout for $T_5$ is

```
> tst.tree3.5
node), split, n, deviance, yval
 * denotes terminal node
1) root 275 802.80 1.0460
   2) phase:dp 226 584.80 1.2020
     4) aspc:exp,ntw 108 247.70 1.5460
       8) lf:bn,bs 83 171.70 1.8330 *
       9) lf:fs,sh,sr,su 25 42.58 0.5933 *
     5) aspc:nte,pro 118 294.00 0.8860
       10) lta:crb 44 99.58 0.4266 *
       11) lta:hil,jeb 74 164.90 1.1590 *
   3) phase:vd 49 150.80 0.3298 *
```

In $T_5$, the split on `phase` partitions the 275 observations into groups of 226 and 49 individuals (node 2 and 3), dependent on whether `phase` is `dp` or `vd`. The latter group is not subdivided further. The former group is then partitioned into groups of 108 and
118 individuals (node 4 and 5), dependent on whether \( \text{aspc} \) is \( \text{(exp, ntw)} \) or is \( \text{(nte, pro)} \). This procedure continues, yielding 5 distinct nodes. And \( T_{12} \) is grown by the same procedure. A complete description of \( T_5 \) and \( T_{12} \) is given in the following table.

<table>
<thead>
<tr>
<th>Trees</th>
<th>Nodes</th>
<th># obs</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>phase = dp, aspc = exp,ntw, If = bn,bs</td>
</tr>
<tr>
<td>( T_5 )</td>
<td>1</td>
<td>83</td>
<td>phase = dp, aspc = exp,ntw, If = fs,sh,sr,su</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>phase = dp, aspc = nte,pro, lta = crb</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>44</td>
<td>phase = dp, aspc = nte,pro, lta = hil,jeb</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>74</td>
<td>phase = dp, aspc = nte,pro, lta = hil,jeb</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>49</td>
<td>phase = vd</td>
</tr>
<tr>
<td>( T_{12} )</td>
<td>1</td>
<td>24</td>
<td>phase = dp, aspc = exp,ntw, If = bn,bs, pos = lm,lo</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>59</td>
<td>phase = dp, aspc = exp,ntw, If = bn,bs, pos = md,um,up</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>25</td>
<td>phase = dp, aspc = nte,pro, lta = crb, geo = cm</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>phase = dp, aspc = nte,pro, lta = crb, geo = lg,ro,ug,vb, aspc = nte</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>phase = dp, aspc = nte,pro, lta = crb, geo = lg,ro,ug,vb, aspc = nte, pos = lm,lo,um</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>phase = dp, aspc = nte,pro, lta = crb, geo = lg,ro,ug,vb, aspc = pro, pos = lm,lo,um</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>15</td>
<td>phase = dp, aspc = nte,pro, lta = crb, geo = lg,ro,ug,vb, aspc = pro, pos = md,up</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>25</td>
<td>phase = dp, aspc = nte,pro, lta = hil,jeb, pos = md,um</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>41</td>
<td>phase = dp, aspc = nte,pro, lta = hil,jeb, pos = lm,lo,up, If = bn,bs,sh,sr</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>8</td>
<td>phase = dp, aspc = nte,pro, lta = hil,jeb, pos = lm,lo,up, If = fs,su</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>31</td>
<td>phase = vd, aspc = exp</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>18</td>
<td>phase = vd, aspc = nte,ntw,pro</td>
</tr>
</tbody>
</table>

Table 1: Terminal nodes for the two candidate trees

Finally, we constructed three sets of charts. In the first set, we produced boxplots of species abundance for each node by species type. By using this chart, we can see how the distributions of dependent variables differ within each node. This tells us that our
tree did a good job in classifying this data set. The second set of charts shows boxplots of abundance by species for each of the five nodes. These charts give another view of how the dependent variables are distributed in the tree. In both of these charts, the node numbers are coded alphabetically ("a" means node 1 so on). The last set of charts gives bar charts for the relative frequencies of levels from each node by all dependent variables.

Discussion of Figures?

Related to statistical decision of
on number of nodes
and statistical validity of results
References


Appendix I. Figures

Figure 1. Cross-validation run 1
Figure 2. Cross-validation run 2
Figure 3. Cross-validation run 3
Figure 4. The full tree for forestry data set
Figure 5. The subtree with 5 nodes for forestry data set
"Subtree with 12 nodes"

Figure 6. The subtree with 12 nodes for forestry data set
Figure 7. Boxplots of abundance by node for each species for the 5 node tree
Figure 8. Boxplots of square root of abundance by species for each node.
Relative Frequencies

Figure 9. Bar charts of abundance class relative frequency by species and node
Appendix II. Splus Program for Multivariate Regression Trees

"mult.tree2" <-
function(formula = formula(data), data = sys.parent(), weights, ywt, subset,
na.action, control = tree.control(nobs, ...), method =
"recursive.partition", model = NULL, x = F, y = T, ...)
{
  if(is.null(model)) {
    model <- match.call(expand = F)
    model$y <- model$x <- model$method <- model$model <- model$control <- model$... <- NULL
    model[[1]] <- as.name("model.frame.default")
    if(!length(model$ywt))
      ywt <- NULL
    else {
      ywt <- eval(model$ywt, sys.parent())
      model$ywt <- NULL
    }
    model <- eval(model, sys.parent())
    if(method == "model.frame")
      return(model)
  }
  Terms <- attr(model, "terms")
  if(any(attr(Terms, "order") > 1))
    stop("Trees cannot handle interaction terms")
  Y <- model.extract(model, "response")  #  if(is.matrix(Y) & ncol(Y) > 1)
  if(attr(Terms, "order") > 1)
    stop("Trees cannot handle multivariate response")
  ylevels <- levels(Y)
  w <- model.extract(model, "weights")  #  if(!length(w))
  if(!length(ywt))
    ywt <- rep(1, nrow(model))
  offset <- attr(Terms, "offset")
  if(!is.null(offset)) {
    if(length(ylevels))
      stop("Cannot use offset with factor response variable")
  }
  #
offset <- model[[offset]]
Y <- Y - offset

X <- tree.matrix(model)
xlevels <- attr(X, "column.levels")
if(is.null(xlevels)) {
    xlevels <- rep(list(NULL), ncol(X))
    names(xlevels) <- dimnames(X)[[2]]
}
assign("xlevels", xlevels, frame = 0)
nobs <- nrow(Y)
if(is.null(control$nobs) && control$nobs < nobs) {
    stop("control$nobs < number of observations in data (user should not set nobs)"
          )
}
if(!is.matrix(X) || !is.matrix(Y))
    stop("X and Y must be matrices")
assign("x0", X, frame = 0)
assign("y0", Y, frame = 0)
assign("splitdat", NULL, frame = 0)
assign("cutpt", NULL, frame = 0)
assign("cuts", NULL, frame = 0)
assign("ncdname", NULL, frame = 0)
assign("where", rep(NA, nrow(X)), frame = 0)
assign("ywt", ywt, frame = 0)
ix <- 1:nrow(X)
dev.null <- dev.calc(ix)
mindev <- control$mindev * dev.null
assign("control", list(mincut = control$mincut, minsize = control$
                      minsize, mindev = mindev), frame = 0)
assign("dev.max", 100 * dev.null, frame = 0)
if(!is.loaded(symbol.For("subs")))
    dyn.load("/home/faculty2/speckman/splus/multcart/code.o")
splits2(ix)
tmp <- splitdat
n <- nrow(splitdat)
frame <- data.frame(splitdat[, 1:4])
names(frame) <- c("var", "n", "dev", "yval")
frame$var <- factor(splitdat[, 1], 0:length(xlevels), c("<leaf>", names(xlevels)))
frame$splits <- cuts
frame$ymeans <- splitdat[, - (1:3)] # added 2/11
row.names(frame) <- nodename[1:n]
fit <- list(frame = frame, where = where, terms = Terms, call =
  match.call())
attr(fit$where, "names") <- row.names(Y)
if(n > 1)
  attr(fit, "class") <- "tree"
else attr(fit, "class") <- "single.node"
attr(fit, "xlevels") <- xlevels
if(length(ylevels))
  attr(fit, "ylevels") <- ylevels
if(x)
  fit$x <- X
if(y)
  fit$y <- Y
fit$ywt <- ywt
fit
"splits2" <-
function(ix, level = 1)
{
  print(level)
  n <- length(ix) # changed the inequality here
  if(n < control$minsize | dev.calc(ix) < control$mindev) {
    # terminal node
    nodedat <- c(0, n, dev.calc(ix), yval.calc(ix))
    assign("splitdat", rbind(splitdat, nodedat), frame = 0)
    assign("cutpt", cbind(cutpt, NA), frame = 0)
    assign("cuts", rbind(cuts, NA), frame = 0)
    assign("nodename", cbind(nodename, level), frame = 0)
    tmp <- where
    tmp[ix] <- length(nodename)
    assign("where", tmp, frame = 0)
  }
}
} else {
  cutdat <- nextcut2(ix)
  if(is.na(cutdat$coord))
    terminal2(ix, level, n)
  else {
    #print(c(cutdat$coord, cutdat$splitpt, n))
    nodedat <- c(cutdat$coord, n, dev.calc(ix), yval.calc(ix))
    assign("splitdat", rbind(splitdat, nodedat), frame = 0)
    assign("cutpt", cbind(cutpt, cutdat$splitpt), frame = 0)
    assign("cuts", rbind(cuts, c(cutdat$cutleft, cutdat$cutright)), frame = 0)
    assign("nodename", cbind(nodename, level), frame = 0)
    splits2(cutdat$left, 2 * level)
    splits2(cutdat$right, 2 * level + 1)
  }
}
"nextcut2"<- function(ix) {
  z <- list(left = NA, right = NA, splitpt = NA, cutleft = NA, cutright = NA)
  coord <- NA
  devtst <- dev.max
  for(k1 in 1:ncol(x0)) {
    z.k <- best3(ix, k1)
    print(c(k1, z.k$splitpt, z.k$devmin, z.k$cutleft, z.k$cutright))
    if(z.k$devmin < devtst) {
      devtst <- z.k$devmin
      z <- z.k
      coord <- k1
    }
  }
}
```r
list(left = z$left, right = z$right, coord = coord, splitpt = z$splitpt,
cutleft = z$cutleft, cutright = z$cutright)
}
"best3"<-function(ix, k1)
{
  n <- length(ix)
devmin <- dev.max
if(!is.null(xlevels[[k1]])) {
  node.levels <- sort(unique(x0[ix, k1]))
p <- length(node.levels)
  if(p == 1)
    list(left = NA, right = NA, devmin = devmin, splitpt =
    NA, cutleft = NA, cutright = NA)
  else {
    ik.dat <- calc.ik(p)[-1, ]
    #for(i in 1:(2^(p - 1) - 1)) {
    #  ik <- ik.dat[i + 1, ]
    #  ik.left <- rep(F, length(ix))
    #  test.levels <- node.levels[ik]
    #  for(j in 1:length(test.levels))
    #    (ik.left <- ik.left | x0[ix, k1] ==
    #     test.levels[j])
    #  ik.right <- !ik.left
    #  ileft <- ix[ik.left]
    #  iright <- ix[ik.right]
    #  if(length(ileft) < control$mincut | length(
    #    iright) < control$mincut)
    #    next
    one <- rep(1, length(ix))
    mat <- outer(node.levels, x0[ix, k1], "==")
    n.l <- ik.dat %*% (mat %*% one)
    n.r <- (lik.dat) %*% (mat %*% one)
    if(length(n.l) == 1) {
      s.l <- as.vector(1/n.l) * ((ik.dat %*% (mat %*%
y0[ix, ]))^2)
      s.r <- as.vector(1/n.r) * (((lik.dat) %*% (mat %*%
```
\[ y_0[ix, j]^2 \]

```r
else {
  s.l <- diag(as.vector(1/n.l)) %*% (ik.dat %*% mat %*% y0[ix, j]^2)
  s.r <- diag(as.vector(1/n.r)) %*% ((ik.dat) %*% (mat %*% y0[ix, j])^2)
}
```

```r
s.lr <- (s.l + s.r) %*% ywt
s.max <- NA
s.matrix <- cbind(c(1:length(s.lr)), s.lr, n.l, n.r)
```

```r
s.order <- s.matrix[order(- s.matrix[, 2]), ]
```

```r
s.order <- matrix(s.order, , 4)
```

```r
for(i in 1:length(s.lr)) {
  if(s.order[i, 3] < control$mincut | s.order[i, 4] < control$mincut)
    next
  else {
    s.max <- s.order[i, 2]
    s.ix <- s.order[i, 1]
    break
  }
}
```

```r
if(s.max == "NA")
  devtst <- dev.max
else {
  devtst <- sum((y0[ix, ] %*% ywt)^2) - s.max
  if(p == 2) {
    test.levels <- node.levels[ik.dat]
    ik <- ik.dat
  }
  else {
    ik <- ik.dat[s.ix, ]
    test.levels <- node.levels[ik]
  }
  ik.left <- rep(F, length(ix))
  for(j in 1:length(test.levels))
```
(ik.left <- ik.left | x0[ix, k1] ==
  test.levels[j])
ik.right <- ik.left
ileft <- ix[ik.left]
iright <- ix[ik.right]
if(devst < devmin) {
  devmin <- devst
  left <- ileft
  right <- iright
  cutleft <- paste(c("", alph[node.levels[ik]]), collapse = "")
  cutright <- paste(c("", alph[node.levels[ik]]), collapse = "")
}
if(devmin == dev.max)
  list(left = NA, right = NA, devmin = devmin,
       splitpt = NA, cutleft = NA, cutright = NA)
else list(left = left, right = right, devmin = devmin,
         splitpt = NA, cutleft = cutleft, cutright =
         cutright)
}
else {
  n <- length(ix)
devmin <- dev.max
ix0 <- ix[order(x0[ix, k1])]
x.ord <- x0[ix0, k1]
valid <- x.ord[-length(x.ord)] != x.ord[-1]
ybar1 <- rep(0, ncol(y0))
ybar2 <- ybar1
rkmat <- ybar1
ileft <- ix0[1:control$mincut]
iright <- ix0[(control$mincut + 1):n]
for(k2 in 1:ncol(y0)) {
  rkmat[k2] <- (var(y0[ileft, k2]) * (length(ileft) - 1) +
                var(y0[iright, k2]) * (length(iright) - 1))
}
42
ybar1[k2] <- mean(y0[ileft, k2])
    ybar2[k2] <- mean(y0[iright, k2])
}
if(valid[control$mmincut]) {
    cutpt <- control$mmincut
    devmin <- sum(rkmat)
}
for(i in (control$mmincut:(n - control$mmincut - 1))) {
    rkmat <- rkmat + i/(i + 1) * ((y0[ix0[i + 1], ] - ybar1)^2 - (n - i)/(n - i - 1) * (y0[ix0[i + 1], ] - ybar2)^2
    ybar1 <- y0[ix0[i + 1], ]/(i + 1) + (ybar1 * i)/(i + 1)
    ybar2 <- (ybar2 * (n - i))/(n - i - 1) - y0[ix0[i + 1], ]/(n - i - 1)
    devtst <- sum(rkmat)
    if(devtst <= devmin & valid[i + 1]) {
        cutpt <- i + 1
        devmin <- devtst
    }
}
if(devmin == dev.max)
    list(left = NA, right = NA, devmin = devmin, splitpt = NA, cutleft = NA, cutright = NA)
else {
    splitpt <- (x0[ix0[cutpt], k1] + x0[ix0[cutpt + 1], k1]) / 2
    cutleft <- paste("<", signif(splitpt, 7), sep = "")
    cutright <- paste(">", signif(splitpt, 7), sep = "")
    list(left = ix0[1:cutpt], right = ix0[ - (1:cutpt)],
    devmin = devmin, splitpt = splitpt, cutleft = cutleft, cutright = cutright)
}
"yval.calc"<- function(ix)
apply(yO[ix, ], 2, mean)

"dev.calc" <- function(ix)
{
  dev <- 0
  df <- length(ix) - 1
  for(k in 1:ncol(yO))
    dev <- dev + var(yO[ix, k]) * ywt[k]
  df * dev
}

"terminal2" <- function(ix, level, n)
{
  nodedat <- c(0, r, dev.calc(ix), yval.calc(ix))
  assign("splitdat", rbind(splitdat, nodedat), frame = 0)
  assign("cutpt", cbind(cutpt, NA), frame = 0)
  assign("nodename", cbind(nodename, level), frame = 0)
  assign("cuts", rbind(cuts, NA), frame = 0)
  tmp <- where
  tmp[ix] <- length(nodename)
  assign("where", tmp, frame = 0)
}

"calc.ik" <- function(p)
{
  ik <- rep(F, p)
  ik.dat <- rbind(ik, rep(F, p))
  for(k in 1:(2^(p - 1) - 1)) {
    i <- 1
    w <- k
    repeat {
      q <- w %% 2
      r <- w %/% 2
      ik[i] <- r == 1
      ik[i + 1] <- q == 1
      i <- i + 1
    }
  }
}
if(q <= 1) {
    ik.dat <- rbind(ik.dat, ik)
    break
}

i <- i + 1
w <- q

ik.dat <- ik.dat[-1, ]

"alph"<- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o")
Appendix III. Splus Program for Application to Problem in Forestry

```splus
code
attach(moddata)
vvre2 <- sqrt(vvre1)
dgrel2 <- sqrt(dgrel)
astepate2 <- sqrt(astepate)
caredigi2 <- sqrt(caredigi)
desmrotu2 <- sqrt(desmrotu)
euphcoro2 <- sqrt(euphcoro)
lespinte2 <- sqrt(lespinte)
monaruss2 <- sqrt(monaruss)
panicomm2 <- sqrt(panicomm)
phrylept2 <- sqrt(phrylept)
smilbona2 <- sqrt(smilbona)
smilrace2 <- sqrt(smilrace)
dep2 <- cbind(vvre2, dgrel2, astepate2, caredigi2, desmrotu2, euphcoro2, lespinte2, monaruss2, panicomm2, phrylept2, smilbona2, smilrace2)
moddata.tree3 <- mult.tree(dep2 ~ lt + geo + If + aspc + phasc + ord + pos)
plot(prune.mult.tree(moddata.tree3))
moddata3.t.30 <- prune.mult.tree(moddata.tree3, best=30)
cv.m3.30 <- cv.mult.tree(moddata3.t.30)
cv2.m3.30 <- cv.mult.tree(moddata3.t.30)
cv3.m3.30 <- cv.mult.tree(moddata3.t.30)
postscript("cv.m3.30.ps")
plot(cv.m3.30)
plot(cv2.m3.30)
plot(cv3.m3.30)
dev.off()

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
For 5 nodes!!
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

tst.tree3.5 <- prune.mult.tree(moddata.tree3, best=5)
post.tree(tst.tree3.5)
where.5 <- tst.tree3.5$where
```
vvrel.list <- list(a=vvrel[where.5==4], b=vvrel[where.5==5], 
c=vvrel[where.5==7], d=vvrel[where.5==8], e=vvrel[where.5==9])
dgrel.list <- list(a=dgrel[where.5==4], b=dgrel[where.5==5], 
c=dgrel[where.5==7], d=dgrel[where.5==8], e=dgrel[where.5==9])
astepate.list <- list(a=astepate[where.5==4], b=astepate[where.5==5], 
c=astepate[where.5==7], d=astepate[where.5==8], e=astepate[where.5==9])
caredigi.list <- list(a=caredigi[where.5==4], b=caredigi[where.5==5], 
c=caredigi[where.5==7], d=caredigi[where.5==8], e=caredigi[where.5==9])
desmrotu.list <- list(a=desmrotu[where.5==4], b=desmrotu[where.5==5], 
c=desmrotu[where.5==7], d=desmrotu[where.5==8], e=desmrotu[where.5==9])
euphcoro.list <- list(a=euphcoro[where.5==4], b=euphcoro[where.5==5], 
c=euphcoro[where.5==7], d=euphcoro[where.5==8], e=euphcoro[where.5==9])
lespinte.list <- list(a=lespinte[where.5==4], b=lespinte[where.5==5], 
c=lespinte[where.5==7], d=lespinte[where.5==8], e=lespinte[where.5==9])
monaruss.list <- list(a=monaruss[where.5==4], b=monaruss[where.5==5], 
c=monaruss[where.5==7], d=monaruss[where.5==8], e=monaruss[where.5==9])
panicomm.list <- list(a=panicomm[where.5==4], b=panicomm[where.5==5], 
c=panicomm[where.5==7], d=panicomm[where.5==8], e=panicomm[where.5==9])
phrylept.list <- list(a=phrylept[where.5==4], b=phrylept[where.5==5], 
c=phrylept[where.5==7], d=phrylept[where.5==8], e=phrylept[where.5==9])
smilbona.list <- list(a=smilbona[where.5==4], b=smilbona[where.5==5], 
c=smilbona[where.5==7], d=smilbona[where.5==8], e=smilbona[where.5==9])
smilrace.list <- list(a=smilrace[where.5==4], b=smilrace[where.5==5], 
c=smilrace[where.5==7], d=smilrace[where.5==8], e=smilrace[where.5==9])

postscript("where.5.ps")
par(mfrow=c(2,2))
boxplot(vvrel.list)
title("vvrel")
boxplot(dgrel.list)
title("dgrel")
boxplot(astepate.list)
title("astepate")
boxplot(caredigi.list)
title("caredigi")
boxplot(desmrotu.list)
title("desmrotu")

47
boxplot(euphcoro.list)
title("euphcoro")
boxplot(lespinte.list)
title("lespinte")
boxplot(monaruss.list)
title("monaruss")
boxplot(panicomm.list)
title("panicomm")
boxplot(phrylept.list)
title("phrylept")
boxplot(smilibona.list)
title("smilibona")
boxplot(smilrace.list)
title("smilrace")
dev.off()

tst.tree3.5.sites <- list(a=sites[where.5==4], b=sites[where.5==5],
c=sites[where.5==7],d=sites[where.5==8],e=sites[where.5==9])

ix.5 <- 1:length(where.5)
sort.5 <- sort(unique(where.5))

nodes.5 <- rep(list(NA),5)
nodes.t <- rep(list(NA),12)

nodes.5.list <- rep(list(nodes.t),5)
for (i in 1:5) nodes.5[[i]] <- ix.5[where.5==sort.5[i]]
for (i in 1:5) {
  for (j in 1:12) {
    nodes.5.list[[i]][[j]] <- dep2[nodes.5[[i]],j]
  }
}

postscript("nodes.5.ps")
par(mfrow=c(3,1))
boxplot(nodes.5.list[[1]],names=names,cex=.5)
title("node 1")
boxplot(nodes.5.list[[2]],names=names,cex=.5)
title("node 2")
boxplot(nodes.5.list[[3]], names=names, cex=.5)
title("node 3")
boxplot(nodes.5.list[[4]], names=names, cex=.5)
title("node 4")
boxplot(nodes.5.list[[5]], names=names, cex=.5)
title("node 5")
dev.off()

-------------------------------------------------------------------------------------

Barchart for 5 nodes!!
-------------------------------------------------------------------------------------

depnames <- c("vvrel", "dgrel", "astepate", "caredigi", "desmrotu", "euphcoro",
"lespinte", "monaruss", "panicomm", "phrylept", "smilbona", "smirrace")
node.ix.5 <- c("node1", "node2", "node3", "node4", "node5")
class.ix <- c("class1", "class2", "class3", "class4", "class5", "class6")
c.spe <- 0
c.node <- 0
c.class <- 0
c.tab <- rep(NA, 12*5*6)
tab.t <- rep(NA, 6)
tab <- 0
add <- 0
for (i in 1:5) {
  for (j in 1:12) {
    tab <- table(ori.dep[nodes.5[[i]], j])/length(ori.dep[nodes.5[[i]], j])*100
    for (k in 1:length(tab)) {
      tab.t[names(tab)[k]==index] <- tab[k]
    }
    c.tab[((j-1)*6+1+add):((j-1)*6+6+add)] <- tab.t
    tab.t <- rep(NA, 6)
  }
  add <- i*72
}
for (i in 1:5) {
  for (j in 1:12) {
    for (k in 1:6) {
      c.class <- c(c.class, class.ix[k])
      c.spe <- c(c.spe, depnames[j])
    }
  }
}
c.node <- c(c.node, node.ix.5[i])
}
)

nodes.5.tab <- data.frame(c.node[-1], c.spe[-1], c.tab, c.class[-1])
names(nodes.5.tab) <- c("node", "species", "freq", "class")

postscript("for.5.ps")
par(mfrow=c(1,1))
barchart(class~freq*node*species, data=nodes.5.tab, xlab="Relative Frequencies", aspect=.4)
dev.off()

# For 12 nodes

tst.tree3.12 <- prune.muti.tree(moddata.tree3, best=12)
post.tree(tst.tree3.12)

where.12 <- tst.tree3.12$where

vvrel.list <- list(a=vvrel[where.12==5], b=vvrel[where.12==6],
c=vvrel[where.12==7], d=vvrel[where.12==10],
e=vvrel[where.12==12],
f=vvrel[where.12==14], g=vvrel[where.12==15],
h=vvrel[where.12==17],
i=vvrel[where.12==19], j=vvrel[where.12==20],
k=vvrel[where.12==22],
l=vvrel[where.12==23])
dgrel.list <- list(a=dgrel[where.12==5], b=dgrel[where.12==6],
c=dgrel[where.12==7], d=dgrel[where.12==10],
e=dgrel[where.12==12],
f=dgrel[where.12==14], g=dgrel[where.12==15],
h=dgrel[where.12==17],
i=dgrel[where.12==19], j=dgrel[where.12==20],
k=dgrel[where.12==22],
l=dgrel[where.12==23])

astepate.list <- list(a=astepate[where.12==5], b=astepate[where.12==6],
c=astepate[where.12==7], d=astepate[where.12==10],
e=astepate[where.12==12],
f=astepate[where.12==14], g=astepate[where.12==15],
h=astepate[where.12==17],
i=astepate[where.12==19], j=astepate[where.12==20],
k=astepate[where.12==22],
l=astepate[where.12==23])
caredigi.list <- list(a=caredigi[where.12==5], b=caredigi[where.12==6],
c=caredigi[where.12==7], d=caredigi[where.12==10],
e=caredigi[where.12==12],
f=caredigi[where.12==14], g=caredigi[where.12==15],
h=caredigi[where.12==17],
i=caredigi[where.12==19], j=caredigi[where.12==20],
k=caredigi[where.12==22],
l=caredigi[where.12==23])
f = caredigi[where.12==14], g = caredigi[where.12==15], h = caredigi[where.12==17],
i = caredigi[where.12==19], j = caredigi[where.12==20], k = caredigi[where.12==22],
l = caredigi[where.12==23])
desmrotu.list <- list(a = desmrotu[where.12==5], b = desmrotu[where.12==6],
c = desmrotu[where.12==7], d = desmrotu[where.12==10], e = desmrotu[where.12==12],
f = desmrotu[where.12==14], g = desmrotu[where.12==15], h = desmrotu[where.12==17],
i = desmrotu[where.12==19], j = desmrotu[where.12==20], k = desmrotu[where.12==22],
l = desmrotu[where.12==23])
euphcoro.list <- list(a = euphcoro[where.12==5], b = euphcoro[where.12==6],
c = euphcoro[where.12==7], d = euphcoro[where.12==10], e = euphcoro[where.12==12],
f = euphcoro[where.12==14], g = euphcoro[where.12==15], h = euphcoro[where.12==17],
i = euphcoro[where.12==19], j = euphcoro[where.12==20], k = euphcoro[where.12==22],
l = euphcoro[where.12==23])
lespinte.list <- list(a = lespinte[where.12==5], b = lespinte[where.12==6],
c = lespinte[where.12==7], d = lespinte[where.12==10], e = lespinte[where.12==12],
f = lespinte[where.12==14], g = lespinte[where.12==15], h = lespinte[where.12==17],
i = lespinte[where.12==19], j = lespinte[where.12==20], k = lespinte[where.12==22],
l = lespinte[where.12==23])
monaruss.list <- list(a = monaruss[where.12==5], b = monaruss[where.12==6],
c = monaruss[where.12==7], d = monaruss[where.12==10], e = monaruss[where.12==12],
f = monaruss[where.12==14], g = monaruss[where.12==15], h = monaruss[where.12==17],
i = monaruss[where.12==19], j = monaruss[where.12==20], k = monaruss[where.12==22],
l = monaruss[where.12==23])
panicomm.list <- list(a = panicomm[where.12==5], b = panicomm[where.12==6],
c = panicomm[where.12==7], d = panicomm[where.12==10], e = panicomm[where.12==12],
f = panicomm[where.12==14], g = panicomm[where.12==15], h = panicomm[where.12==17],
i = panicomm[where.12==19], j = panicomm[where.12==20], k = panicomm[where.12==22],
l = panicomm[where.12==23])
phrylept.list <- list(a = phrylept[where.12==5], b = phrylept[where.12==6],
c = phrylept[where.12==7], d = phrylept[where.12==10], e = phrylept[where.12==12],
f = phrylept[where.12==14], g = phrylept[where.12==15], h = phrylept[where.12==17],
i = phrylept[where.12==19], j = phrylept[where.12==20], k = phrylept[where.12==22],
l = phrylept[where.12==23])
smilbona.list <- list(a = smilbona[where.12==5], b = smilbona[where.12==6],
c = smilbona[where.12==7], d = smilbona[where.12==10], e = smilbona[where.12==12],
f = smilbona[where.12==14], g = smilbona[where.12==15], h = smilbona[where.12==17],
i = smilbona[where.12==19], j = smilbona[where.12==20], k = smilbona[where.12==22],
l = smilbona[where.12==23])
l=smilbona[where.12==23])
smirrace.list <- list(a=smirrace[where.12==5], b=smirrace[where.12==6],
c=smirrace[where.12==7], d=smirrace[where.12==10], e=smirrace[where.12==12],
f=smirrace[where.12==14], g=smirrace[where.12==15], h=smirrace[where.12==17],
i=smirrace[where.12==19], j=smirrace[where.12==20], k=smirrace[where.12==22],
l=smirrace[where.12==23])

postscript("where.12.ps")
par(mfrow=c(2,2))
boxplot(vwrel.list)
title("vwrel")
boxplot(dgrel.list)
title("dgrel")
boxplot(astepate.list)
title("astepate")
boxplot(caredigi.list)
title("caredigi")
boxplot(desmrotu.list)
title("desmrotu")
boxplot(euphcoro.list)
title("euphcoro")
boxplot(iespinte.list)
title("iespinte")
boxplot(monoruss.list)
title("monoruss")
boxplot(panicomm.list)
title("panicomm")
boxplot(phrylept.list)
title("phrylept")
boxplot(smilbona.list)
title("smilbona")
boxplot(smirrace.list)
title("smirrace")
dev.off()

tst.tree3.12.sites <- list(a=sites[where.12==5], b=sites[where.12==6],
c=sites[where.12==7], d=sites[where.12==10], e=sites[where.12==12],
c=52
\begin{verbatim}
f=sites[where.12==14], g=sites[where.12==15], h=sites[where.12==17], 
i=sites[where.12==19], j=sites[where.12==20], k=sites[where.12==22], 
l=sites[where.12==23])

ix.12 <- 1:length(where.12)
sort.12 <- sort(unique(where.12))
nodes.12 <- rep(list(NA),12)
nodes.t <- rep(list(NA),12)
nodes.12.list <- rep(list(nodes.12),12)
for (i in 1:12) nodes.12[[i]] <- ix.12[where.12==sort.12[i]]
for (i in 1:12) {
  for (j in 1:12) {
    nodes.12.list[[i]][[j]] <- dep2[nodes.12[[i]],j]
  }
}

postscript("nodes.12.ps")
par(mfrow=c(3,1))
boxplot(nodes.12.list[[1]],names=names,cex=.5)
title("node 1")
boxplot(nodes.12.list[[2]],names=names,cex=.5)
title("node 2")
boxplot(nodes.12.list[[3]],names=names,cex=.5)
title("node 3")
boxplot(nodes.12.list[[4]],names=names,cex=.5)
title("node 4")
boxplot(nodes.12.list[[5]],names=names,cex=.5)
title("node 5")
boxplot(nodes.12.list[[6]],names=names,cex=.5)
title("node 6")
boxplot(nodes.12.list[[7]],names=names,cex=.5)
title("node 7")
boxplot(nodes.12.list[[8]],names=names,cex=.5)
title("node 8")
boxplot(nodes.12.list[[9]],names=names,cex=.5)
title("node 9")
boxplot(nodes.12.list[[10]],names=names,cex=.5)
\end{verbatim}
title("node 10")
boxplot(nodes.12.list[[11]],names=names,cex=.5)
title("node 11")
boxplot(nodes.12.list[[12]],names=names,cex=.5)
title("node 12")
dev.off()

Barchart for 12 nodes!!

node.ix.12 <- c("node1", "node2", "node3", "node4", "node5", "node6", "node7", "node8",
"node9", "node10", "node11", "node12")
index <- c("0", "0.01", "0.5", "3", "10", "20")
class.ix <- c("class1", "class2", "class3", "class4", "class5", "class6")
c.spe <- 0
c.node <- 0
c.class <- 0
c.tab <- rep(0, 6*12*6)
tab.t <- rep(0, 6)
tab <- 0
add <- 0
for (i in 1:6) {
  for (j in 1:12) {
    tab <- table(ori.dep[nodes.12[[i]],j])/length(ori.dep[nodes.12[[i]],j])*100
    for (k in 1:length(tab)) {
      tab.t[3ames(tab)[k]==index] <- tab[k]
    }
    c.tab[((j-1)*6+1+add)+(j-1)*6+6+add)] <- tab.t
  }
  add <- add+72
}
for (i in 1:6) {
  for (j in 1:12) {
    for (k in 1:6) {
      c.class <- c(c.class, class.ix[k])
      c.spe <- c(c.spe, depnames[j])
      c.node <- c(c.node, node.ix.12[i])
    }
  }
}
nodes.12.tab.1 <- data.frame(c.node[-1],c.spe[-1],c.tab,c.class[-1])
names(nodes.12.tab.1) <- c("node","species","freq","class")

c.spe <- 0
c.node <- 0
c.class <- 0
c.tab <- rep(0,6*12*6)
tab.t <- rep(0,6)
tab <- 0
add <- 0
for (i in 1:6) {
  for (j in 1:12) {
    tab <- table(ori.dep[nodes.12[(i+6)],j]) / length(ori.dep[nodes.12[(i+6)],j]) * 100
    for (k in 1:length(tab)) {
      tab.t[names(tab)[k] == index] <- tab[k]
    }
    c.tab[((j-1)*6+1+add):((j-1)*6+6+add)] <- tab.t
    tab.t <- rep(0,6)
  }
  add <- i*72
}
for (i in 7:12) {
  for (j in 1:12) {
    for (k in 1:6) {
      c.class <- c(c.class, class.ix[k])
      c.spe <- c(c.spe, depnames[j])
      c.node <- c(c.node, node.ix.12[i])
    }
  }
}

nodes.12.tab.2 <- data.frame(c.node[-1],c.spe[-1],c.tab,c.class[-1])
names(nodes.12.tab.2) <- c("node","species","freq","class")
postscript("for.12.ps")
par(mfrow=c(1,1))
barchart(class~freqnode*species,data=nodes.12.tab.1,xlab="Relative Frequencies",aspect=.4)
barchart(class~freqnode*species,data=nodes.12.tab.2,xlab="Relative Frequencies",aspect=.4)
dev.off()