Journal of Statistical Software
February 2018, Volume 83, Issue 8. doi: 10.18637/jss.v083.i08

PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees

Eun-Kyung Lee
Ewha Womans University

Abstract

PPtreeViz, an R package, was developed to explore projection pursuit methods for classification. It provides functions to calculate various projection pursuit indices for classification and to explore the results in the space of projection. It also provides functions for the projection pursuit classification tree. The visualization methods of the tree structure and the features of each node in **PPtreeViz** can be used to easily explore the projection pursuit classification tree structure and determine the characteristics of each class. To calculate the projection pursuit indices and optimize these indices, we use the **Rcpp** and **RcppArmadillo** packages in R to improve the speed.

Keywords: R, classification tree, projection pursuit, exploratory data analysis, data visualization.

1. Introduction

Projection pursuit (Huber 1985) uses the projection pursuit index and optimization procedure to find an interesting low dimensional projection. Such an interesting feature is defined by the projection pursuit index, and we usually maximize the predefined projection pursuit index to find an interesting projection. An interesting projection for classification is a view with the most separable classes. Several projection pursuit indices with class information have been suggested. For example, Lee, Cook, Klinke, and Lumley (2005) proposed the LDA (linear discriminant analysis) projection pursuit index using class information through an extension of the linear discriminant analysis idea. They also proposed the L_p index. These indices work for a reasonable amount of data with a small number of variables. The PDA (penalized discriminant analysis) index was developed for highly correlated data or a small number of observations with very large number of variables (Lee and Cook 2010).

Lee, Cook, Park, and Lee (2013) proposed the projection pursuit classification tree, a new approach to build a classification tree using projection pursuit indices with class information.

At each node, the projection pursuit classification tree uses the best projection to separate two groups of classes using various projection pursuit indices with class information. One class is assigned to only one final node and the depth of the projection pursuit classification tree cannot be greater than the number of classes. Therefore, the projection pursuit classification tree constructs a simple but more understandable tree for classification. The projection coefficients of each node represent the importance of the variables to the class separation of each node. The behaviors of these coefficients are useful to explore how classes are separated in a tree.

PPtreeViz is an R (R Core Team 2017) package that was developed to explore the projection pursuit indices with classification as well as the projection pursuit classification tree. It provides an initial overview of the data for classification, allowing us to further explore the entire tree structure as well as each node of the tree.

A couple of R packages have been developed to draw a graphical representation of the classification tree. The generic function `plot.rpart` from package **rpart** (Therneau, Atkinson, and Ripley 2017) shows a basic representation of the tree structure, and the `prp` and `fancyRpartPlot` functions in **rpart.plot** (Milborrow 2017) package show more fancy plots. Packages **party** (Hothorn, Hornik, Strobl, and Zeileis 2017) and **partykit** (Hothorn and Zeileis 2015) include plots of the tree with various options of the inner terminal node for the user to explore the tree structure. Also, the tree representation is neat and clear, and each inner node has an id number that can be used for further analysis. Most of these packages focus on the representation of the tree structure and the status of the classes in each node. We adapted the way that **party** draws the tree structure to represent the projection pursuit classification tree and then added more features to explore the data space of each node. **PPtreeViz** also provides the visualization tools to explore projection pursuit indices with class information. This package is a useful exploratory data analysis tool for classification.

In Section 2, we review the various projection pursuit indices with class information, propose two new indices with class information, and describe optimization methods for projection pursuit indices. Visual exploration methods for the projection pursuit are also discussed. The usages of R functions to calculate and optimize index values, and the method used to visualize the results are described. In Section 3, we outline the way the projection pursuit classification tree is constructed and show how to fit the projection pursuit classification tree with functions in **PPtreeViz**. The visualization methods used to explore the projection pursuit classification tree structure are discussed in Section 4. Examples and discussion then follow.

2. Projection pursuit for classification

2.1. Projection pursuit indices with class information

Let $\mathbf{X}_{ij}$ be the p-dimensional vector of the j-th observation in the i-th class, $i = 1, \ldots, g$, $j = 1, \ldots, n_i$, g is the number of classes, and n_i is the number of observations in class i. Also, let

$$\begin{aligned}
\mathbf{B}_i &= (\bar{\mathbf{X}}_{i.} - \bar{\mathbf{X}}_{..})(\bar{\mathbf{X}}_{i.} - \bar{\mathbf{X}}_{..})^\top, \qquad (1)\\
\mathbf{W}_i &= \frac{1}{n_i}\sum_{j=1}^{n_i}(\mathbf{X}_{ij} - \bar{\mathbf{X}}_{i.})(\mathbf{X}_{ij} - \bar{\mathbf{X}}_{i.})^\top,
\end{aligned}$$

where $\bar{\mathbf{X}}_{i.} = \sum_{j=1}^{n_i} \mathbf{X}_{ij}/n_i$ is the mean of the i-th class, $\bar{\mathbf{X}}_{..} = \sum_{i=1}^{g} \sum_{j=1}^{n_i} \mathbf{X}_{ij}/n$ is the overall mean, and $n = \sum_{i=1}^{g} n_i$ is the total number of observations.

LDA index

$$I_{\text{LDA,W}}(\mathbf{A}) = 1 - \frac{|\mathbf{A}^\top \mathbf{W} \mathbf{A}|}{|\mathbf{A}^\top (\mathbf{W} + \mathbf{B}) \mathbf{A}|} \tag{2}$$

where $\mathbf{B} = \sum_{i=1}^{g} n_i \mathbf{B}_i$, $\mathbf{W} = \sum_{i=1}^{g} n_i \mathbf{W}_i$ and $\mathbf{A}$ is a projection matrix.

In this index, n_i is used as the weight of class i. If the sizes of classes (n_i) differ substantially, the result of the projection pursuit method is affected by the sizes of the classes, and is usually dominated by the class with a large size. Even though the observed data have different class sizes, it might be interesting to find separations that are not influenced by class size. For this purpose, we developed indices with the same weights, n/g, and this is equivalent to using no weight.

$$I_{\text{LDA,noW}}(\mathbf{A}) = 1 - \frac{|\mathbf{A}^\top \mathbf{W}^* \mathbf{A}|}{|\mathbf{A}^\top (\mathbf{W}^* + \mathbf{B}^*) \mathbf{A}|} \tag{3}$$

where $\mathbf{B}^* = \sum_{i=1}^{g} \mathbf{B}_i$ and $\mathbf{W}^* = \sum_{i=1}^{g} \mathbf{W}_i$.

The `LDAindex` function is developed to calculate the LDA index value. It is written in C++ using the **Rcpp** (Eddelbuettel and François 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014) packages in R. `origclass` is the group information; it should be a vector of integers, characters or factors. `origdata` is the original data without group information and should be a matrix. `proj` is the projection vector to be used if the user wants to project the original data with a default value of `NULL`. If the user does not provide his/her own projection vector, `LDAindex` treats `origdata` as the projected data and calculates the LDA index value of the original data. `weight` is the option for the weight in the index calculation and the default value is `TRUE`.

```
R> LDAindex(origclass = iris[, 5],
+    origdata = as.matrix(iris[, 1:4]))

[1] 0.9765614

R> LDAindex(origclass = iris[, 5],
+    origdata = as.matrix(iris[, 1, drop = FALSE]))
R> LDAindex(origclass = iris[, 5], origdata = as.matrix(iris[, 1:2]),
+    proj = matrix(c(1, 0), nrow = 2))

[1] 0.6187057
```

PDA index

When variables are highly correlated, $|\mathbf{A}^\top (\mathbf{W} + \mathbf{B}) \mathbf{A}|$ in the LDA index is close to zero and the LDA index does not work properly. The PDA index (Lee and Cook 2010) has a penalty term with λ to prevent the value of the determinant from being close to zero. λ is the value in $[0,1]$ and controls the proportion of the penalty term. If $\lambda = 0$, no penalty term is added and the PDA index is the same as the LDA index. If $\lambda = 1$, all variables are treated as uncorrelated variables. If the observed data are highly correlated, we need to use a large λ.

The definition of the PDA index in Lee and Cook (2010) only works for the standardized data. We extend this idea to the raw data. Let

$$\mathbf{W}_{\mathrm{PDA}} = \mathrm{diag}(\mathbf{W}) + (1-\lambda)\mathrm{offdiag}(\mathbf{W}) \tag{4}$$

where

$$\begin{aligned} \mathrm{diag}(\mathbf{W}) &= \mathrm{diag}(w_{11}, \ldots, w_{pp}) \\ \mathrm{offdiag}(\mathbf{W}) &= \mathbf{W} - \mathrm{diag}(\mathbf{W}) \end{aligned} \tag{5}$$

Then,

$$I_{\mathrm{PDA,W}}(\mathbf{A}, \lambda) = 1 - \frac{|\mathbf{A}^\top \mathbf{W}_{\mathrm{PDA}} \mathbf{A}|}{|\mathbf{A}^\top (\mathbf{W}_{\mathrm{PDA}} + \mathbf{B})\, \mathbf{A}|}. \tag{6}$$

The main idea of the PDA index is to keep the diagonal elements of $\mathbf{W}$ and to reduce the effect of the off-diagonal elements using λ. This approach weakens the correlations among variables and keeps the variances of the variables. The `PDAindex` function is developed to calculate the PDA index value, and it has one extra argument, `lambda`. The default value of `lambda` is 0.1.

```
R> PDAindex(origclass = iris[, 5], origdata = as.matrix(iris[, 1:2]),
+    lambda = 0.2)

[1] 0.8198746
```

L_r *index*

The L_r index ignores the correlations among the variables, focusing only on the variations of each variable, and uses L_r measure to calculate the variations. With various r, we can find different views with separable classes (Lee *et al.* 2005). In **PPtreeViz**, we modified the definition of L_r index to keep the index value in $[0,1]$.

Let

$$\mathbf{Y}_{ij,proj} = \mathbf{A}^\top * \mathbf{X}_{ij} = \left[\begin{array}{ccc} y_{ij1} & \cdots & y_{ijq} \end{array}\right]^\top \tag{7}$$

be the projected data of $\mathbf{X}_{ij}$ onto the q-dimensional projection $\mathbf{A}$ and

$$B_{ri} = \sum_{l=1}^{q} |\bar{y}_{i.l} - \bar{y}_{..l}|^r \tag{8}$$

$$W_{ri} = \frac{1}{n_i}\sum_{l=1}^{q}\sum_{j=1}^{n_i} |y_{ijl} - \bar{y}_{i.l}|^r \tag{9}$$

Then,

$$I_{\mathrm{Lr,W}}(\mathbf{A}) = 1 - \frac{(W_r)^{1/r}}{(W_r + B_r)^{1/r}} \tag{10}$$

where $\bar{y}_{i.l} = \sum_{j=1}^{n_i} y_{ijl}/n_i$, $\bar{y}_{..l} = \sum_{i=1}^{g}\sum_{j=1}^{n_i} y_{ijl}/n$, $B_r = \sum_{i=1}^{g} n_i B_{ri}$, and $W_r = \sum_{i=1}^{g} n_i W_{ri}$. We can also use the same weight n/g instead of n_i.

$$I_{\mathrm{Lr,noW}}(\mathbf{A}) = 1 - \frac{({W_r}^*)^{1/r}}{({W_r}^* + {B_r}^*)^{1/r}} \tag{11}$$

where ${B_r}^* = \sum_{i=1}^{g} \frac{n}{g} B_{ri}$, ${W_r}^* = \sum_{i=1}^{g} \frac{n}{g} W_{ri}$.
The `Lrindex` function is developed to calculate the L_r index value. `r` should be a positive integer value and the default value is 1.

```
R> Lrindex(origclass = iris[, 5], origdata = as.matrix(iris[, 1:2]), r = 1)
```

```
[1] 0.5497106
```

1D Gini index

The Gini is the impurity measure used in the classification tree (Hastie, Friedman, and Tibshirani 2011). Since this measure is defined for one-dimensional variables, the Gini projection pursuit index is developed to find a one-dimensional projection only. Let $y_{ij,proj} = \mathbf{a}^\top \mathbf{X}_{ij}$ and $y_1^* \leq \cdots \leq y_n^*$ be the ordered observations of $y_{ij,proj}$, $i = 1, \ldots, g$ and $j = 1, \ldots, n_i$. For each $k = 1, \ldots, n-1$, let

$$\begin{aligned} p_{Li,k} &= \frac{\#\text{ of observations of class } i \text{ in } \{y_1^*, \ldots, y_k^*\}}{k} \\ p_{Ri,k} &= \frac{\#\text{ of observations of class } i \text{ in } \left\{y_{k+1}^*, \ldots, y_n^*\right\}}{n-k}, \end{aligned} \tag{12}$$

and

$$\begin{aligned} P_{L,\mathrm{Gini},k} &= -\sum_{i=1}^{g} p_{Li,k}(1 - p_{Li,k}) \\ P_{R,\mathrm{Gini},k} &= -\sum_{i=1}^{g} p_{Ri,k}(1 - p_{Ri,k}) \\ P_{\mathrm{Gini},k} &= \left(\frac{k}{n}\right) P_{L,\mathrm{Gini},k} + \left(\frac{n-k}{n}\right) P_{R,\mathrm{Gini},k}. \end{aligned} \tag{13}$$

Then,

$$I_{\mathrm{Gini}}(\mathbf{a}) = \max_{k=1,\ldots,n-1} \{(g-1) + g * P_{\mathrm{Gini},k}\}$$

where $\frac{1-g}{g} \leq P_{\mathrm{Gini},k} \leq \frac{2-g}{g}$ and $P_{\mathrm{Gini},k} = \frac{2-g}{g}$ when all observations in $\{y_1^*, \ldots, y_k^*\}$ or $\left\{y_{k+1}^*, \ldots, y_n^*\right\}$ are in the same class. The `GINIindex1D` function calculates the Gini index value.

```
R> GINIindex1D(origclass = iris[, 5],
+    origdata = as.matrix(iris[, 1, drop = FALSE]))

[1] 0.683281
```

1D entropy index

The entropy is another impurity measure used in the classification tree (Hastie *et al.* 2011). We use the same approach to the 1D Gini index. Let

$$\begin{aligned} P_{L,\text{Entropy},k} &= \sum_{i=1}^{g} p_{Li,k} \log p_{Li,k} & (14) \\ P_{R,\text{Entropy},k} &= \sum_{i=1}^{g} p_{Ri,k} \log p_{Ri,k} \\ P_{\text{Entropy},k} &= \left(\frac{k}{n}\right) P_{L,\text{Entropy},k} + \left(\frac{n-k}{n}\right) P_{R,\text{Entropy},k}. \end{aligned}$$

Then,

$$I_{\text{Entropy}}(\mathbf{a}) = \max_{k=1,\ldots,n-1} \left\{ \frac{1 + P_{\text{Entropy},k}/\log(g)}{1 + \text{maxE}/\log(g)} \right\} \qquad (15)$$

where $-\log(g) \leq P_{\text{Entropy},k} \leq \text{maxE}$ and

$$\text{maxE} = \begin{cases} \log 2 - \log g & \text{if } g \text{ is an even number} \\ -\frac{1}{2}\log\left(\frac{g^2-1}{4}\right) + \frac{1}{2g}\log\left(\frac{g-1}{g+1}\right) & \text{if } g \text{ is an odd number} \end{cases}$$

The property of the entropy index differs from that of the Gini index. When g is an even number, $P_{\text{Entropy},k}$ has the value maxE when all observations in $\{y_1^*, \ldots, y_k^*\}$ are in g/2 classes and $\left\{y_{k+1}^*, \ldots, y_n^*\right\}$ are in the other g/2 classes. This means that the entropy index has the maximum value when two groups of classes are separable and each group consists of the same number of classes. When g is an odd number, $P_{\text{Entropy},k}$ has the value maxE when all observations in $\{y_1^*, \ldots, y_k^*\}$ are in the (g+1)/2 classes and $\left\{y_{k+1}^*, \ldots, y_n^*\right\}$ are in the other classes. Therefore, the entropy index finds different views from those of the Gini index. The `ENTROPYindex1D` function is developed to calculate the entropy index value.

```
R> ENTROPYindex1D(origclass = iris[, 5],
+    origdata = as.matrix(iris[, 1, drop = FALSE]))

[1] 0.6068117
```

2.2. Optimization for the projection pursuit with LDA and PDA indices

We need to use an optimization procedure to find an interesting projection using the projection pursuit index. For the LDA and PDA indices, we can calculate the theoretical optimum using the maximization lemma (Johnson and Wichern 2007). For the other indices, we use the simulated annealing optimization method.

LDA index optimization

Finding the optimum $\mathbf{A}$ to maximize $I_{\text{LDA,W}}(\mathbf{A})$ is the same as finding $\mathbf{A}$ to maximize the following

$$\frac{|\mathbf{A}^\top \mathbf{B}\mathbf{A}|}{|\mathbf{A}^\top (\mathbf{W}+\mathbf{B})\,\mathbf{A}|}. \tag{16}$$

Therefore, the optimal q-dimensional projection to maximize $I_{\text{LDA,W}}(\mathbf{A})$ is the first q eigenvectors of $(\mathbf{W}+\mathbf{B})^{-1}\,\mathbf{B}$. The same approach can be applied to $I_{\text{LDA,noW}}(\mathbf{A})$.

The `LDAopt` function is designed to find the theoretical optimal projection to maximize the LDA index with various dimension of the projection. To calculate $(\mathbf{W}+\mathbf{B})^{-1}$, we use the `ginv` function in the **MASS** package (Ripley 2017; Venables and Ripley 2002). `q` represents the dimension of the projection.

```
R> LDA.proj.result <- LDAopt(origclass = iris[, 5],
+    origdata = iris[, 1:4], q = 2)
R> LDA.proj.result

$indexbest
[1] 0.9765614

$projbest
           [,1]         [,2]
[1,] -0.2087418  0.006531964
[2,] -0.3862037  0.586610553
[3,]  0.5540117 -0.252561540
[4,]  0.7073504  0.769453092
```

PDA index optimization

Finding the optimum $\mathbf{A}$ to maximize $I_{\text{PDA,W}}(\mathbf{A})$ is the same as finding $\mathbf{A}$ to maximize the following

$$\frac{|\mathbf{A}^\top \mathbf{B}\mathbf{A}|}{|\mathbf{A}^\top (\mathbf{W}_{PDA}+\mathbf{B})\,\mathbf{A}|}. \tag{17}$$

Therefore the optimal q-dimensional projection to maximize $I_{\text{PDA,W}}(\mathbf{A})$ is the first q eigenvectors of $(\mathbf{W}_{PDA}+\mathbf{B})^{-1}\,\mathbf{B}$. The `PDAopt` function is designed to find the theoretical optimal projection to maximize the PDA index.

```
R> PDA.proj.result <- PDAopt(origclass = iris[, 5], origdata = iris[, 1:4],
+    weight = TRUE, q = 1, lambda = 0.1)
R> PDA.proj.result
```

```
$indexbest
[1] 0.9696339

$projbest
           [,1]
[1,] -0.1440983
[2,] -0.3964776
[3,]  0.5001605
[4,]  0.7562280
```

2.3. Simulated annealing optimization

Since it is not easy to find the theoretical optimum projection for the L_r, Gini and entropy indices, we modified the simulated annealing optimization algorithm to fit the projection pursuit method. This algorithm can be applied to general projection pursuit indices. The first approach for the simulated annealing optimization for the projection pursuit is found in Lee *et al.* (2005). We modified their algorithm slightly to find the global optimum more quickly and more precisely. We also give guidelines to set up the initial values of the parameters.

Simulated annealing optimization algorithm for the projection pursuit

```
1. Set an initial projection $\mathbf{A}_0$ and $temp = 1$.
2. Calculate the projection pursuit index value $I_0 = I(A_0)$.
3. For $k = 1, 2, \ldots,$ maxiter,
      step 1   $temp = temp * \texttt{cooling}$.
               For fast optimization, use $temp = temp * \texttt{cooling}^2$ while $k < 1000$.
      step 2   $\mathbf{A}_k = temp *$ random projection $+ \mathbf{A}_0$.
      step 3   Calculate $I_k = I(\mathbf{A}_k)$ and $d.index = I_k - I_0$
      step 4   If $d.index > 0$, $\mathbf{A}_0 \leftarrow \mathbf{A}_k$ and $I_0 \leftarrow I_k$.
               else $\mathbf{A}_0 \leftarrow \mathbf{A}_k$ and $I_0 \leftarrow I_k$ with probability $\exp(d.index/tempp)$
               where $tempp = \texttt{energy}/\log(\texttt{k}+2)$.
   Repeat step1 1 to 4 until $|d.index| < TOL$ or $k =$ maxiter, where $TOL = \texttt{energy}/10^6$
   .
```

Several parameters are used in this simulated annealing optimization. The `cooling` parameter is used to determine the step size of a new projection $\mathbf{A}_k$. This parameter is in $(0, 1)$. If we use a small value for `cooling`, the step size is rapidly reduced, and it becomes difficult to find the global optimum. If we use `cooling` close to 1, the step size is slowly reduced, but there is a strong chance to find the global optimum. We recommend using 0.999 for our projection pursuit indices for classification to find the global optimum.

The `energy` parameter is used for the probability to take new projection. The smaller `energy`, the higher the probability and the chance to take a new projection will be high. However, the probability is also determined by the difference between the new projection pursuit index and the old projection pursuit index, and the range of these values depends on the data. We suggest that `energy` $= 1 - I$ where I is the projection pursuit index of the original data. The `PPopt` function is designed to find the optimal projection to maximize the Lr, Gini and entropy indices.

```
R> PP.proj.result <- PPopt(origclass = iris[, 5],
+    origdata = as.matrix(iris[, 1:4]), PPmethod = "Lr", r = 1)
R> PP.proj.result

$indexbest
[1] 0.8665575

$projbest
           [,1]
[1,] -0.1332361
[2,] -0.4391126
[3,]  0.5394665
[4,]  0.7059775
```

2.4. Explore the projection pursuit indices with the two-dimensional data

To explore the properties of various projection pursuit indices with the two-dimensional data, we provide an R function for Huber's plot (Huber 1990). This plot represents the projection pursuit index values in all possible directions in a 2D space. These indices are calculated using the projections for $(\cos\theta, \sin\theta)$, $\theta = 1^\circ, \ldots, 180^\circ$. For each projection, we calculate the projected data and the index value of the projected data. The circle with a dashed line represents the median of all index values and the solid circle shows the other index values relative to this median. The projection with the maximum index value is indicated as a solid line and the histogram of the data projected on to this optimal projection is also provided. We use the **ggplot2** package (Wickham 2009) to draw Huber plot and the histogram and use **gridExtra** package (Auguie and Antonov 2015) to arrange these two plots. Figure 1 shows a simple example of the Huber's plot of the LDA index with two variables from iris data - the sepal length and the sepal width. From the histogram of the projected data on the best projection, we can see the separation of the setosa class from the other two classes, except for one observation in the setosa.

```
R> Huberplot(as.matrix(iris[, 1:2]), iris[, 5], PPmethod = "LDA")
```

The `Huberplot` function provides `PPmethod` options for various projection pursuit indices with class information. Also the user can use his/her own function with `PPmethod = "UserDef"` and `UserDefFtn` options to calculate the index.

For the simple example, we use the principal component analysis. To find the first principal component, we need to maximize the variance of the projected data, therefore we can define to `sampleIndex` as a function to calculate the variance of the projected data. Figure 2 shows the result of `UserDefFtn = sampleIndex`.

```
R> sampleIndex <- function(proj.data){
+    var(proj.data)
+  }
```

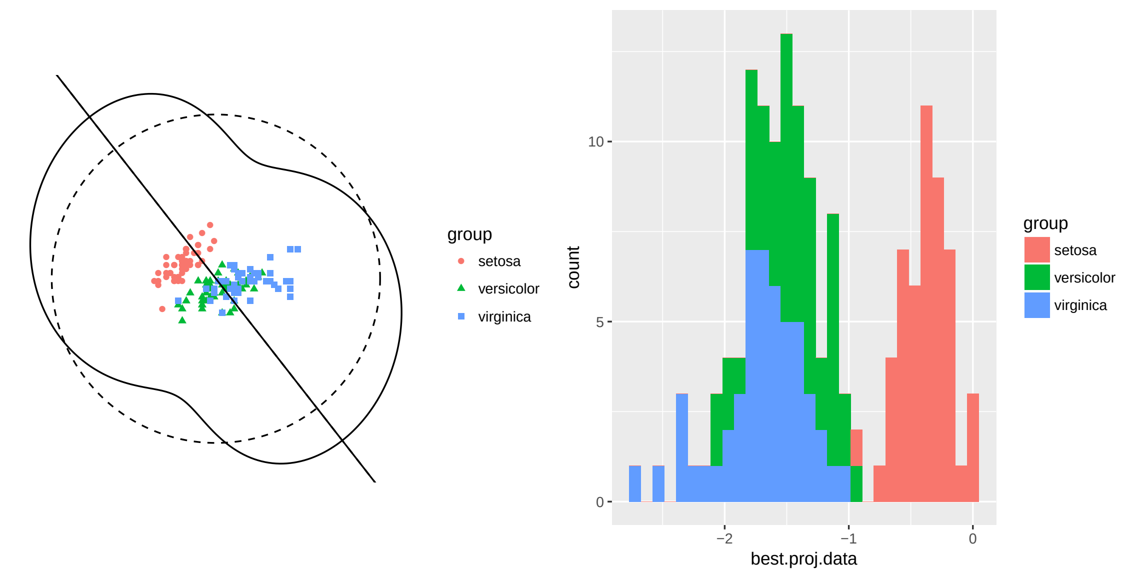


Figure 1: Huber's plot with the sepal length and the sepal width in the iris data. LDA index with `weight = TRUE` is used.

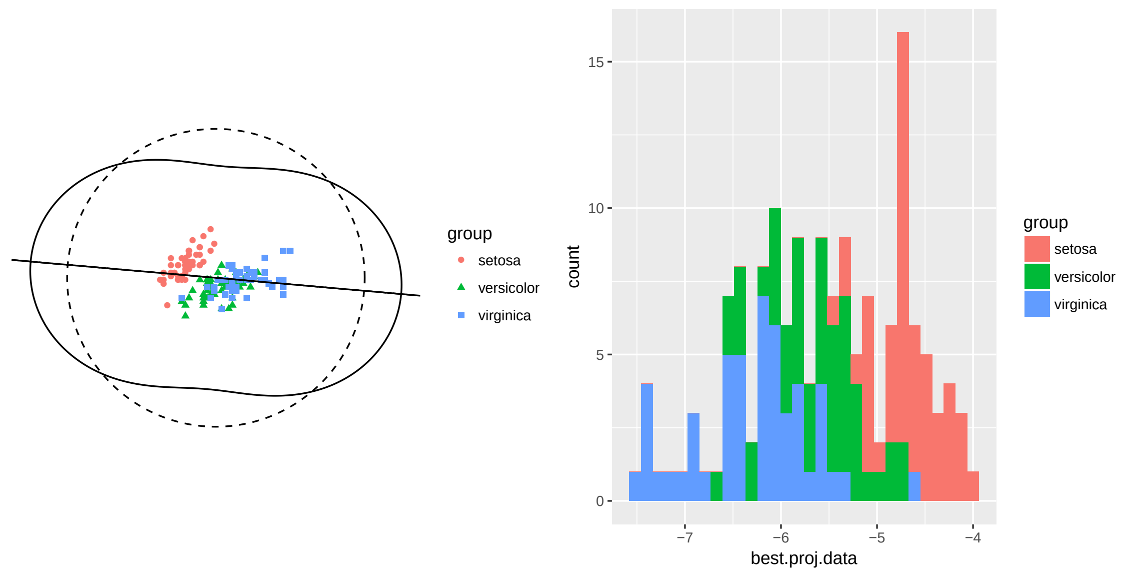


Figure 2: Huber's plot for the first principal component. The user defined index function `sampleIndex` is used.

```
R> Huberplot(iris[, 1:2], iris[, 5], PPmethod = "UserDef",
+    UserDefFtn = sampleIndex)
```

2.5. Exploring the optimal projection for classification

After finding the optimal projection in the q-dimensional space, it is worthwhile to explore the data projected onto the optimal projection with the projection coefficients. We developed

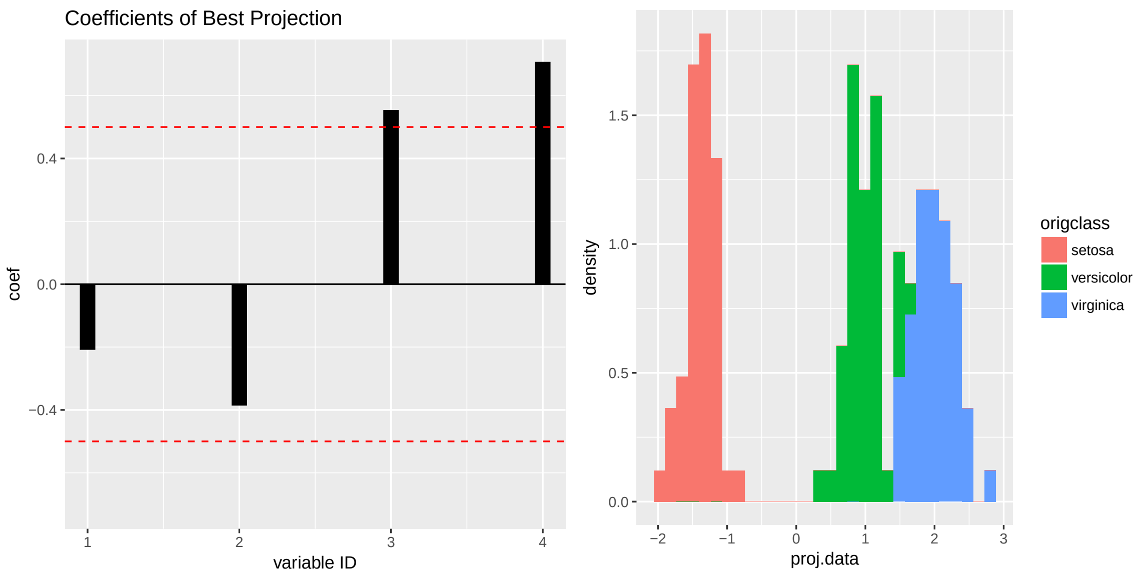


Figure 3: 1D `LDAopt` result with iris data.

the `PPoptViz` function to explore the projection space. The plot for the coefficient values of the projection and the histogram of the projected data are provided for $q = 1$. The coefficient values for each variable are represented using bars from zero (represented as a black line) and the red dotted lines at $\pm 1/\sqrt{p}$ represent the guidelines showing the significant importance of the coefficients. If the coefficient value is outside of these dotted lines, we can conclude that the corresponding variable plays an important role in the separation. The histogram represents the distributions of each class with different colors and helps detect the view with separable classes. We then use the plot of the projection coefficient values, to determine which variables are important to separate classes in this optimal view.

Figure 3 shows the best projection coefficients and data projected onto the best projection with iris data using the LDA index. Variables 3 and 4 have more important roles than the other two variables to separate the setosa class. We also can observe some separation between virginica and versicolor with some overlap between the two classes.

```
R> PPoptViz(LDAopt(origclass = iris[,5], origdata = iris[,1:4], q = 1))
```

For $q > 1$, the plots are provided with the type of $q * q$ matrix. The diagonal parts represent the coefficient plots of the optimal projection in each dimension and the off-diagonal parts represent the scatter plot of the data projected onto the corresponding projection. Figure 4 shows the best 2D projection coefficients and a scatter plot of the projected data. In the scatter plot for the 1st dimension (dim 1) and 2nd dimension (dim 2), we can clearly see that the setosa class is separated from the other classes. We also can see that the versicolor and virginica class appear to be separable, although this is not clear and we cannot observe this separation without color. Even though variables 2 and 4 have larger projection coefficients than the other two variables in the 2nd dimension, this 2nd projection does not have an important role for class separation. For iris data, the 1st projection shows enough separation.

```
R> PPoptViz(LDAopt(origclass = iris[, 5], origdata = iris[, 1:4], q = 2))
```

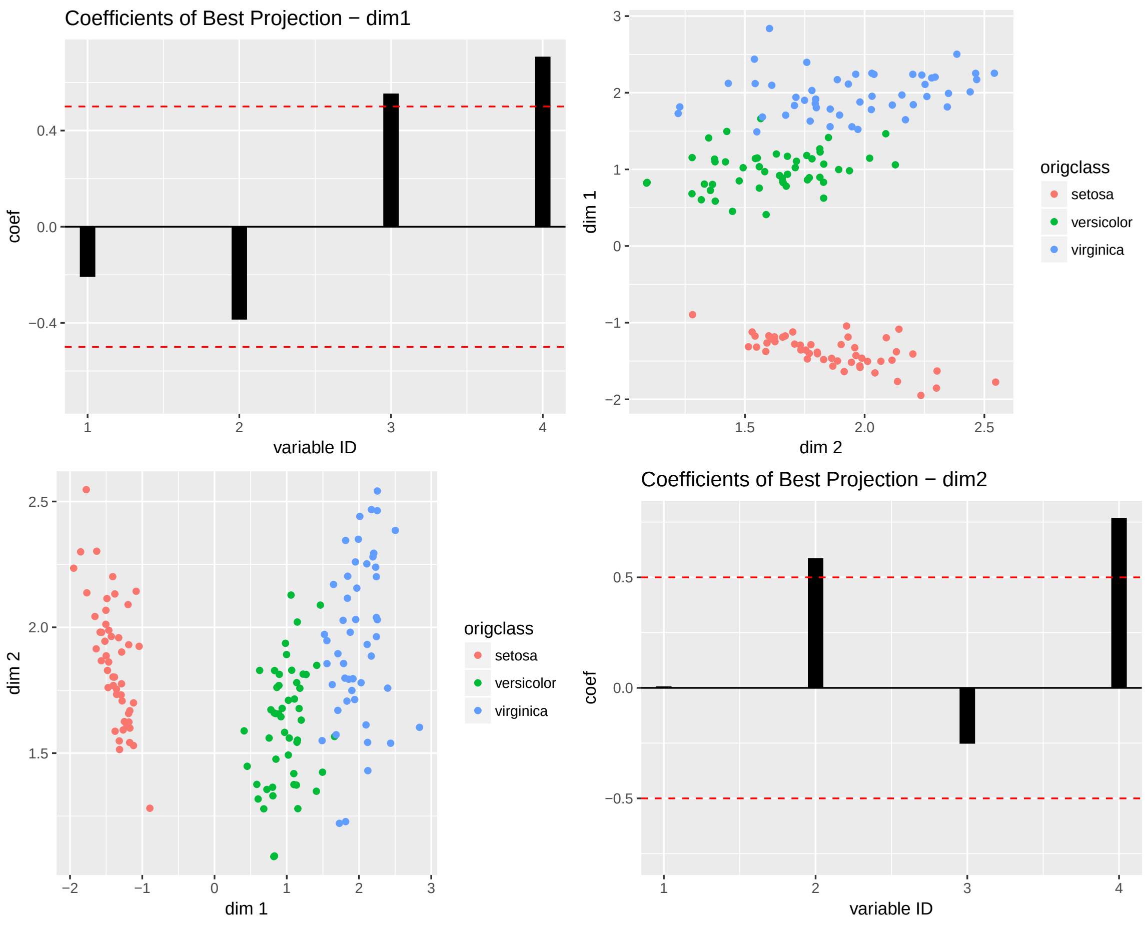


Figure 4: 2D `LDAopt` result with iris data.

3. Projection pursuit classification tree

In this chapter, we explore how to construct the projection pursuit classification tree using **PPtreeViz**. A projection pursuit classification tree (Lee *et al.* 2013) is a type of classification tree that uses projection pursuit indices with class information. The usual tree-structured classification finds the rule used to separate data into two groups using impurity measures that determine the degree of purity of two groups in terms of classes. In contrast, the projection pursuit classification tree finds the rule used to separate classes into two groups. This rule uses the best projection to separate two groups of classes with various projection pursuit indices with class information. One class is assigned to only one final node and the maximum depth of the projection pursuit classification tree is the number of classes. Therefore, the projection pursuit classification tree constructs a simple but more understandable tree for classification. The projection coefficients for each node represent the importance of the variable in separating the classes in each node, and the behaviors of these coefficients are useful to explore how classes are separable.

3.1. Projection pursuit classification tree with various indices

The original package **PPtree** uses C to calculate the projection pursuit indices and to optimize these indices. For the projection pursuit classification tree, the **PPtree** package also provides three different functions with different indices and optimization methods: `LDA.Tree`, `PDA.Tree`, and `PP.Tree`. At present, **Rcpp** is a popular tool to develop R packages with a heavy computational load. Therefore we want to move on to use **Rcpp** instead of C. We combine these three functions into the `PPtreeClass` for the projection pursuit classification tree with the `PPmethod` option in **PPtreeViz** library. To use this function, we need at least the two arguments, `formula` and `data`. The default value of `PPmethod` is “`LDA"`. In the projection pursuit classification tree, we provide 8 different rules to define the cutoff values for each node. Let m_1, med_1, s_1, IQR_1, and n_1 be the mean, median, standard deviation, inter quartile range, and sample size of the left group at each node, respectively, and let m_2, med_2, s_2, IQR_2, and n_2 be the mean, median, standard deviation, inter quartile range, and sample size of the right group. The rules are then defined using the following formula:

$$\begin{aligned}
\text{Rule 1} &= 0.5 * m_1 + 0.5 * m_2 \qquad (18)\\
\text{Rule 2} &= \left(\frac{n_2}{n_1+n_2}\right) * m_1 + \left(\frac{n_1}{n_1+n_2}\right) * m_2\\
\text{Rule 3} &= \left(\frac{s_2}{s_1+s_2}\right) * m_1 + \frac{s_1}{s_1+s_2} * m_2\\
\text{Rule 4} &= \left(\frac{s_2/\sqrt{n_2}}{s_1/\sqrt{n_1}+s_2/\sqrt{n_2}}\right) * m_1 + \left(\frac{s_1/\sqrt{n_1}}{s_1/\sqrt{n_1}+s_2/\sqrt{n_2}}\right) * m_2\\
\text{Rule 5} &= 0.5 * med_1 + 0.5 * med_2\\
\text{Rule 6} &= \left(\frac{n_2}{n_1+n_2}\right) * med_1 + \left(\frac{n_1}{n_1+n_2}\right) * med_2\\
\text{Rule 7} &= \left(\frac{IQR_2}{IQR_1+IQR_2}\right) * med_1 + \left(\frac{IQR_1}{IQR_1+IQR_2}\right) * med_2\\
\text{Rule 8} &= \left(\frac{IQR_2/\sqrt{n_2}}{IQR_1/\sqrt{n_1}+IQR_2/\sqrt{n_2}}\right) * med_1 + \left(\frac{IQR_1/\sqrt{n_1}}{IQR_1/\sqrt{n_1}+IQR_2/\sqrt{n_2}}\right) * med_2
\end{aligned}$$

To calculate the means, medians, inter quartile ranges and standard deviations in each rule, we use the **stats** packages (R Core Team 2017).

```
R> Tree.result <- PPTreeclass(Species~.,data = iris, PPmethod = "LDA")
R> Tree.result

=============================================================
Projection Pursuit Classification Tree result
=============================================================

1) root
   2)* proj1*X < cut1  ->  "setosa"
   3)  proj1*X >= cut1
      4)* proj2*X < cut2  ->  "versicolor"
      5)* proj2*X >= cut2  ->  "virginica"
```

```
Error rates of various cutoff values
-------------------------------------------------------------
           Rule1 Rule2  Rule3  Rule4 Rule5 Rule6  Rule7  Rule8
error.rate  0.02  0.02 0.0267 0.0267  0.02  0.02 0.0267 0.0267
```

`print` is a generic plot function for the projection pursuit classification tree object, which is the result from `PPtreeClass`. It prints the result of the projection pursuit classification tree. The simple tree structure and the error rates with each rule are printed without any other options. To print the projection coefficients and cutoff values at each node, we use `coef.print = TRUE` and `cutoff.print = TRUE` options.

```
R> print(Tree.result, coef.print = TRUE, cutoff.print = TRUE)

=============================================================
Projection Pursuit Classification Tree result
=============================================================

1) root
   2)* proj1*X < cut1  ->  "setosa"
   3)  proj1*X >= cut1
      4)* proj2*X < cut2  ->  "versicolor"
      5)* proj2*X >= cut2  ->  "virginica"

Projection Coefficient in each node
-------------------------------------------------------------
      1:"Sepal.Length" 2:"Sepal.Width" 3:"Petal.Length" 4:"Petal.Width"
proj1          -0.1929         -0.7096           0.6565          0.1679
proj2          -0.2268         -0.3558           0.4446          0.7901

Cutoff values of each node
-------------------------------------------------------------
       Rule1   Rule2   Rule3   Rule4   Rule5   Rule6   Rule7   Rule8
cut1 -1.0708 -1.5130 -1.2795 -1.0513 -1.0633 -1.4964 -1.3141 -1.0921
cut2  1.0629  1.0629  1.0444  1.0444  1.0589  1.0589  0.9882  0.9882

Error rates of various cutoff values
-------------------------------------------------------------
           Rule1 Rule2  Rule3  Rule4 Rule5 Rule6  Rule7  Rule8
error.rate  0.02  0.02 0.0267 0.0267  0.02  0.02 0.0267 0.0267
```

After fitting the projection pursuit classification tree, we can predict classes for new observations using the `predict` function.

```
R> n <- nrow(iris)
R> train.id <- sample(c(1:n), n * 0.9)
R> test.id <- c(1:n)[-train.id]
R> Tree.train <- PPTreeclass(Species ~ ., data = iris[train.id, ],
+    PPmethod = "LDA")
```

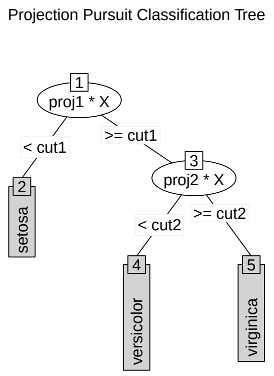
```
R> predict(Tree.train, iris[train.id, 1:4], Rule = 1)

  [1] versicolor setosa     setosa     versicolor virginica
  [6] virginica  virginica  versicolor versicolor setosa
...
[131] setosa     virginica  virginica  setosa     virginica
Levels: setosa versicolor virginica

R> predict(Tree.train, iris[test.id, 1:4], Rule = 1)

 [1] setosa     setosa     setosa     setosa     setosa
 [6] versicolor versicolor versicolor versicolor virginica
[11] versicolor virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

We also provide the `PPclassify` function. The `PPclassify` can count the number of prediction errors if we know about the true class information.

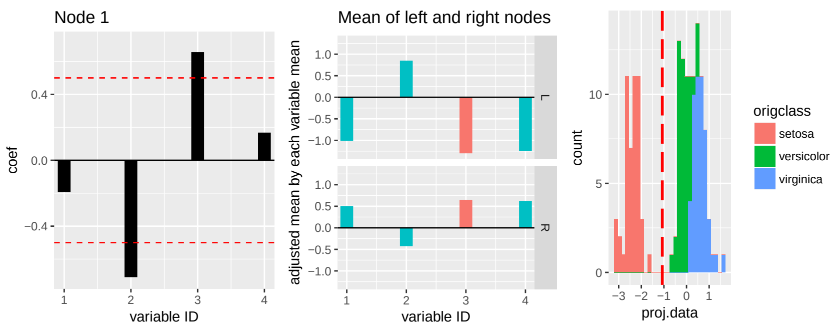
```
R> PPclassify(Tree.result = Tree.train, test.data = iris[train.id, 1:4],
+    Rule = 1, true.class = iris[train.id, 5])

$predict.error
[1] 2

$predict.class
  [1] "versicolor" "setosa"     "setosa"     "versicolor" "virginica"
  [6] "virginica"  "virginica"  "versicolor" "versicolor" "setosa"
  . . .
[131] "setosa"     "virginica"  "virginica"  "setosa"     "virginica"

R> PPclassify(Tree.result = Tree.train, test.data = iris[test.id, 1:4],
+    Rule = 1, true.class = iris[test.id, 5])

$predict.error
[1] 1

$predict.class
 [1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
 [6] "versicolor" "versicolor" "versicolor" "versicolor" "virginica"
[11] "versicolor" "virginica"  "virginica"  "virginica"  "virginica"
```

4. Visualization of projection pursuit classification tree

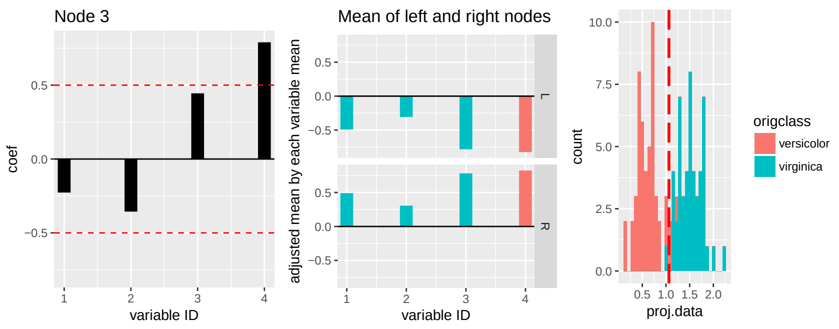
The projection pursuit classification tree focuses on the exploratory analysis as well as the precision of classification. With this tree structure, we can determine the variables that play



(a) Projection pursuit classification tree.



(b) Node 1.



(c) Node 3.

Figure 5: Plot of the structure of projection pursuit classification tree with iris data and plots of nodes 1 and 3.

important roles in each separation. Thus, it is very important to explore each node in the projection pursuit classification tree structure and to find out how to divide classes into two groups. This is the main advantage of the projection pursuit classification tree. Therefore we need to develop tools to explore the projection space of each node. **PPtreeViz** provides two functions to explore the projection pursuit tree - `plot` and `PPclassNodeViz`. For the plot with the tree structure, we modified `plot` for `BinaryTree` in the **party** package (Hothorn *et al.* 2017). The `plot` function in the **party** package shows inner nodes with node id and edges with condition. For the final nodes, however, it represents summary statistics for the classification results. For the projection pursuit classification tree, we need to show the name of the class and the node id for the final node. Therefore we need to modify `plot` in **party**. The `plot` function in **PPtreeViz** is a generic plot function for an object with `PPtreeclass` class, which is the result from the `PPtreeClass` function. The `font.size` and `width.size` options are available for a large tree with a large number of groups. `font.size` determines the size of the letters in the plot, and its default value is 17. `width.size` determines the size of the ellipse for the inner node, and its default value is 1.

```
R> plot(Tree.result)
R> PPclassNodeViz(Tree.result, node.id = 1, Rule = 1)
R> PPclassNodeViz(Tree.result, node.id = 3, Rule = 1)
```

Figure 5(a) represents the projection pursuit classification tree with iris data. The ellipses represent the inner nodes and the rectangles with group names represent the final nodes. The node id is shown in the square box by each node. In each ellipse, we find the optimal 1D projection using the LDA index and find the cutoff values with various rules. At node 1, the projection pursuit classification tree separates “setosa” from the other two classes and at node 3, this tree separates “virginica” from “versicolor”. We can explore these inner nodes more closely using the `PPclassNodeViz` function, and the result is shown in Figure 5(b). We can use the plot of the projection coefficients to figure out that variables 2 and 3 have important roles in separating “setosa” from the other two groups, but they work in a different direction. This means that “setosa” has a large value for variable 2 and a small value for variable 3. In contrast, the other two classes have a small value of variable 2 and a large value of variable 3. This result is shown in the middle plot titled “Mean of left and right nodes”. In this plot, “L” refers to the left group (“setosa” in node 1) and “R” refers to the right group (“versicolor” and “verginica”).

The dashed vertical red line in the histogram of the projected data represents the cutoff values for the specified rule (-1.0708 in this plot). These coefficient values and cutoff values can be checked with the `coef.print = TRUE` and `cutoff.print = TRUE` options in `print` function. No errors occur at node 1.

Figure 5(c) shows the result for node 3 of the tree. From the coefficient plot, we can see that variable 4 is the most important variable to separate “virginica” from “versicolor”. In this node, a few observations are misclassified with rule 1.

The projection coefficients at each node are related to the correlations among variables. Therefore it is helpful to explore the coefficients using the correlations. In the `PPclassNodeViz` function, we provide an `image` option to draw the image plot of the correlation matrix. The default value is `FALSE`, and in the next chapter we show how to use `image = TRUE` through examples.

In the **PPtreeViz** library, we define the `PPtreeclass` class to save all results from fitting the `PPtreeClass` function. Currently, **partykit** is commonly used to summarize and visualize tree structure in various ways. To use the properties of **partykit**, we define the `as.party.PPtreeclass` function to add `PPtreeclass` class to the `party` class.

```
R> Party.result <- as.party(Tree.result)
R> Party.result

Model formula:
Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width

Fitted party:
[1] root
|   [2] proj1 >= -1.07081
|   |   [3] proj2 >= 1.06291: virginica (n = 51, err = 3.9%)
|   |   [4] proj2 < 1.06291: versicolor (n = 49, err = 2.0%)
|   [5] proj1 < -1.07081: setosa (n = 50, err = 0.0%)

Number of inner nodes:    2
Number of terminal nodes: 3

R> plot(Party.result)
```

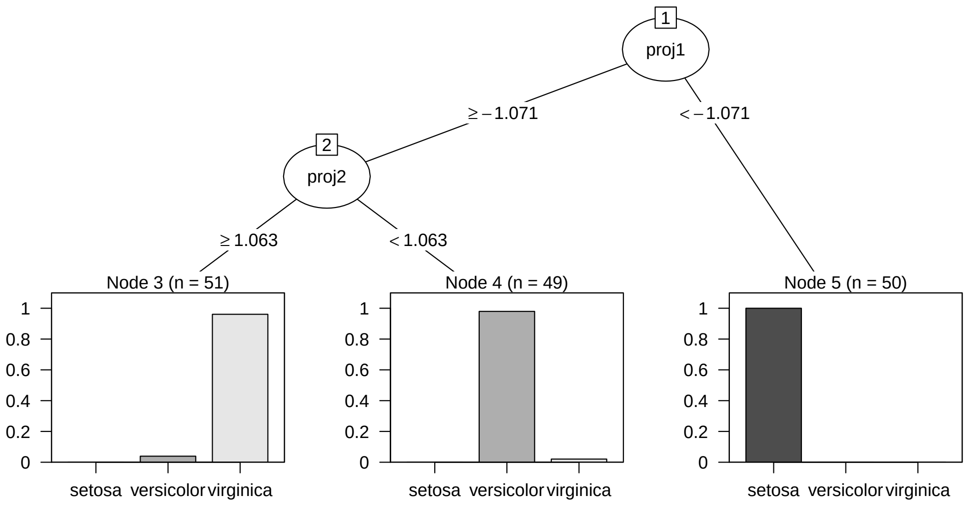


Figure 6: Plot of the structure of projection pursuit classification tree with **partykit**.

V1	Weight of the fish
V2	Body length – length from the nose to the beginning of the tail
V3	Tail length – length from the beginning of the tail to the end of the tail
V4	Ratio of the length from the notch of the tail to the end of the tail to the tail length
V5	Ratio of height to the total length
V6	Ratio of width to the total length

Table 1: Variables in fishcatch data.

5. Example

5.1. Fishcatch data

This data set is from the Journal of Statistical Education Data Archive. It contains information on 159 fish from 7 species (Bream, Parkki, Perch, Pike, Roach, Smelt, and Whitewish). Six continuous variables (weight, length1, length2, length3, height and width) and one binary variable (sex) are used. We modified the variables and make 6 new continuous variables (Table 1). We use these 6 variables for the 7 species to explain how to use **PPtreeViz** to explore the projection pursuit classification tree and each node of the tree. The result of the fitted projection pursuit classification tree with PDA index ($\lambda = 0.1$) is:

```
=============================================================
Projection Pursuit Classification Tree result
=============================================================

1) root
   2)  proj1*X < cut1
      4)  proj2*X < cut2
         6)* proj3*X < cut3  ->  "Pike"
```

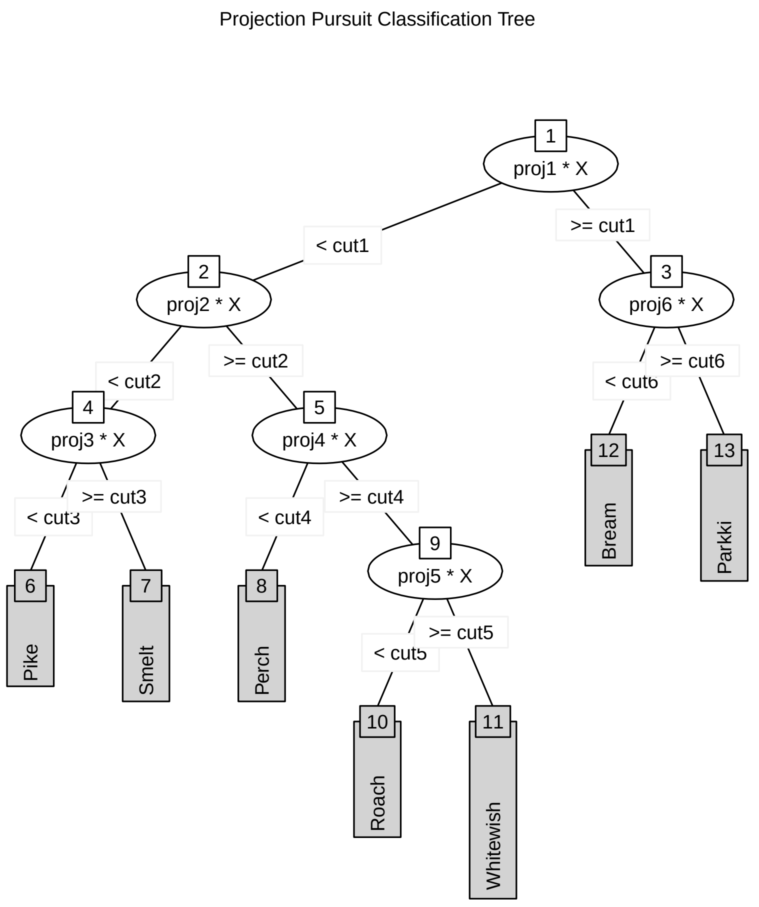


Figure 7: Projection pursuit classification tree with fishery data.

```
         7)* proj3*X >= cut3  ->  "Smelt"
      5)  proj2*X >= cut2
         8)* proj4*X < cut4  ->  "Perch"
         9)  proj4*X >= cut4
            10)* proj5*X < cut5  ->  "Roach"
            11)* proj5*X >= cut5  ->  "Whitewish"
   3)  proj1*X >= cut1
      12)* proj6*X < cut6  ->  "Bream"
      13)* proj6*X >= cut6  ->  "Parkki"

Cutoff values of each node
-------------------------------------------------------------
       Rule1   Rule2   Rule3   Rule4   Rule5   Rule6   Rule7   Rule8
cut1  0.4239  0.8478  0.5452  0.3201  0.4122  0.8593  0.4955  0.2584
cut2 -0.8744 -1.2294 -0.8974 -0.7087 -0.9088 -1.2668 -1.1193 -0.9316
cut3 -0.3148 -0.2130 -0.0232 -0.0709 -0.3069 -0.2052 -0.0286 -0.0766
cut4 -0.4208 -0.1629 -0.4567 -0.5887 -0.4127 -0.1737 -0.4410 -0.5636
cut5 -0.3622 -0.0113 -0.3868 -0.5749 -0.4100 -0.0738 -0.5108 -0.6805
cut6 -0.7299 -0.3221 -0.8085 -1.0203 -0.7229 -0.3042 -0.8778 -1.0841
```

```
Error rates of various cutoff values
-------------------------------------------------------------
           Rule1  Rule2 Rule3  Rule4  Rule5  Rule6  Rule7  Rule8
error.rate     0 0.0252     0 0.0126 0.0063 0.0314 0.0126 0.0252
```

Figure 7 is the result of the projection pursuit classification tree using the PDA index with `lambda=0.1`. No misclassification occurs in this projection pursuit classification tree with rules 1 and 3. At node 1, we can separate Bream and Parkki from the other fish classes. Figure 8 represents the plot of node 1 with the `image = TRUE` option. In the histogram of the projected data (the upper right panel), we can clearly see that Parkki and Bream are separated from the other fish classes. The coefficient plot (the upper left panel) shows that the height (V5) is the most important variable. From this plot, we can see that Parkki and Bream have a tall (V5) and narrow width (V6) relative to the other classes of fish. In the correlation plot (the lower right panel), yellow represents positively correlated and blue represents negatively correlated variables. From this image plot, we can see that V1, V2 and V3 are positively correlated, V4 and V6 are negatively correlated, and V5 and V6 are uncorrelated.

At node 2 (Figure 9), the projection pursuit classification tree separates Pike and Smelt from the other fish classes. In this separation, the ratio of height (V5) is important. Pike and Smelt are shorter than the other fish classes. In the data of node 2, V4 is negatively correlated with V5 and V6. In contrast to the data at node 1, V5 and V6 are positively correlated at node 2.

Bream and Parkki are clearly separated in node 3, and Figure 10 presents the result of node 3. In this separation, the tail length (V3) is the most important variable while the ratio of the notch length (V4) also plays an important role. This result reveals that Bream has a much longer tail and much deeper notch than Pakki.

Pike and Smelt are separated at node 4 (Figure 11). Pike has longer body (V2), longer tail (V3), and lighter weight (V1) than Smelt. In the data of node 4, V1, V2, and V3 are highly negatively correlated with V4. Perch is separated from the others at node 5. From Figure 12, we can see that Perch has a shorter tail (V3) and a longer length (V2), with a small notch in the tail (V4) relative to Roach and Whitewish. At node 9 separation (Figure 13), we can separate Roach from Whitewish with the role of two important variables, the ratio of the notch (V4) and the weight (V1) of the fish. Whitewish are heavier than Roach and have a small notch in the tail compared to that for Roach. The data at node 9 show that V4 is negatively correlated with all other variables.

The projection pursuit classification tree can be used in **PPtreeViz** to determine the special characteristics of each species, especially from a classification point of view. Fortunately, there are no misclassification cases in this example. However, we can notice any misclassification and the reason for it at each node by using the histogram of the projected data and the values of the projection coefficients.

5.2. Isolated letter speech recognition data

The isolated letter recognition data (ISOLET) is from the UCI Machine Learning repository (`http://archive.ics.uci.edu/ml`). In this repository, two data sets are available: one for training and the other for testing. The training set has 240 observations and the testing set

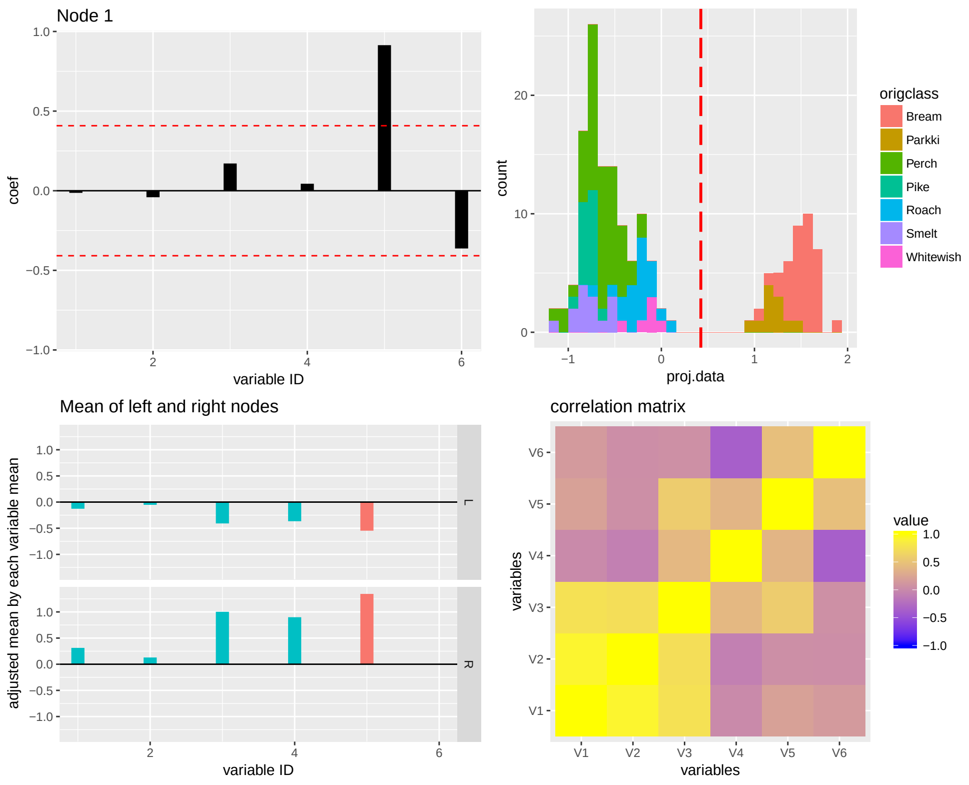


Figure 8: Node 1 of fishery data.

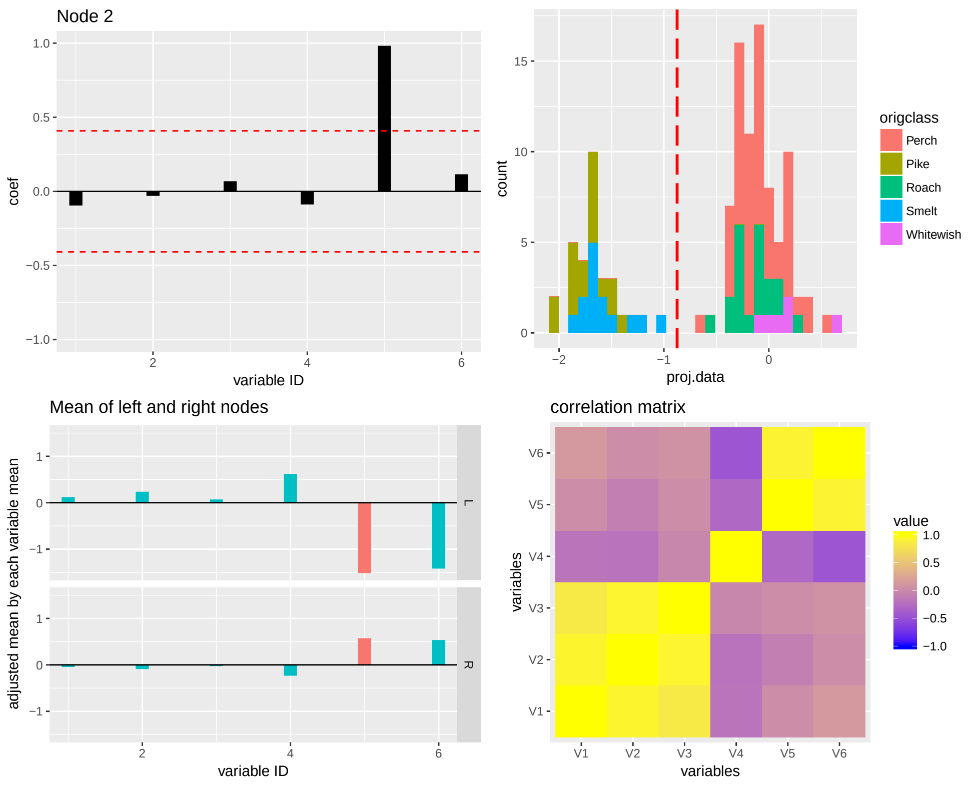


Figure 9: Node 2 of fishery data.

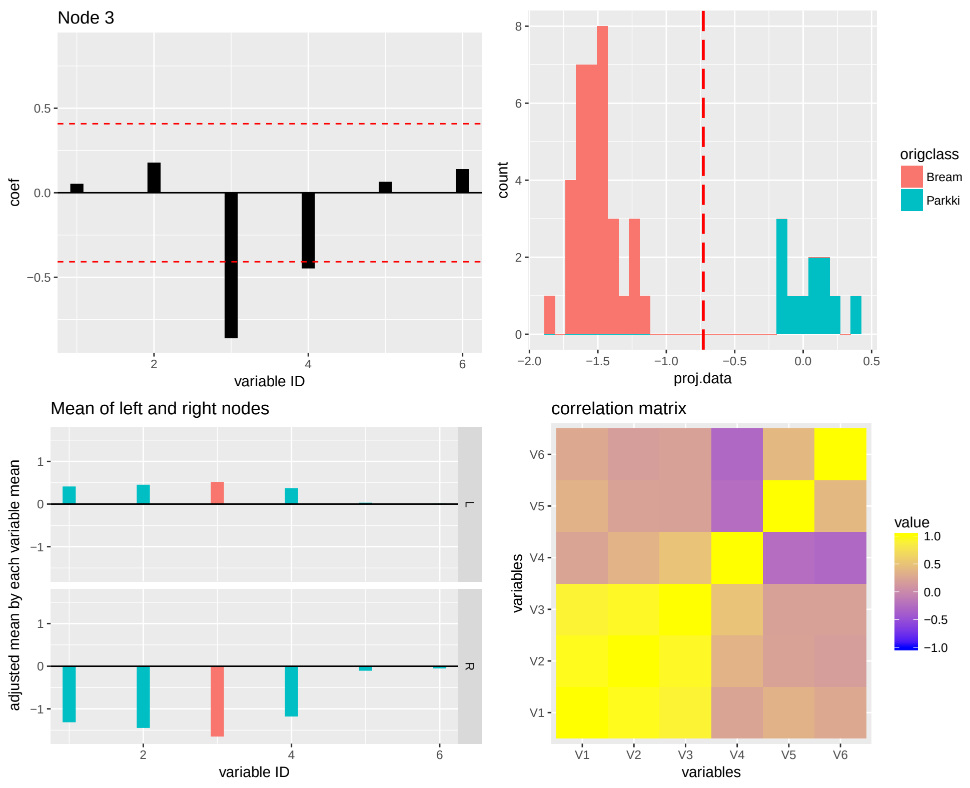


Figure 10: Node 3 of fishery data.

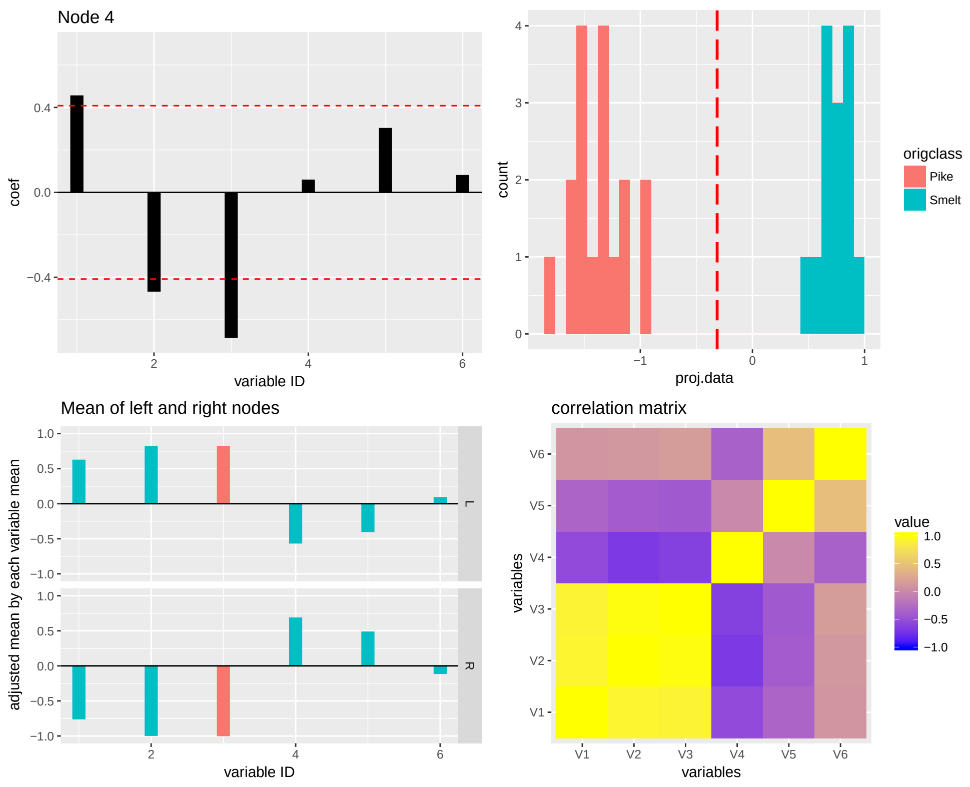


Figure 11: Node 4 of fishery data.

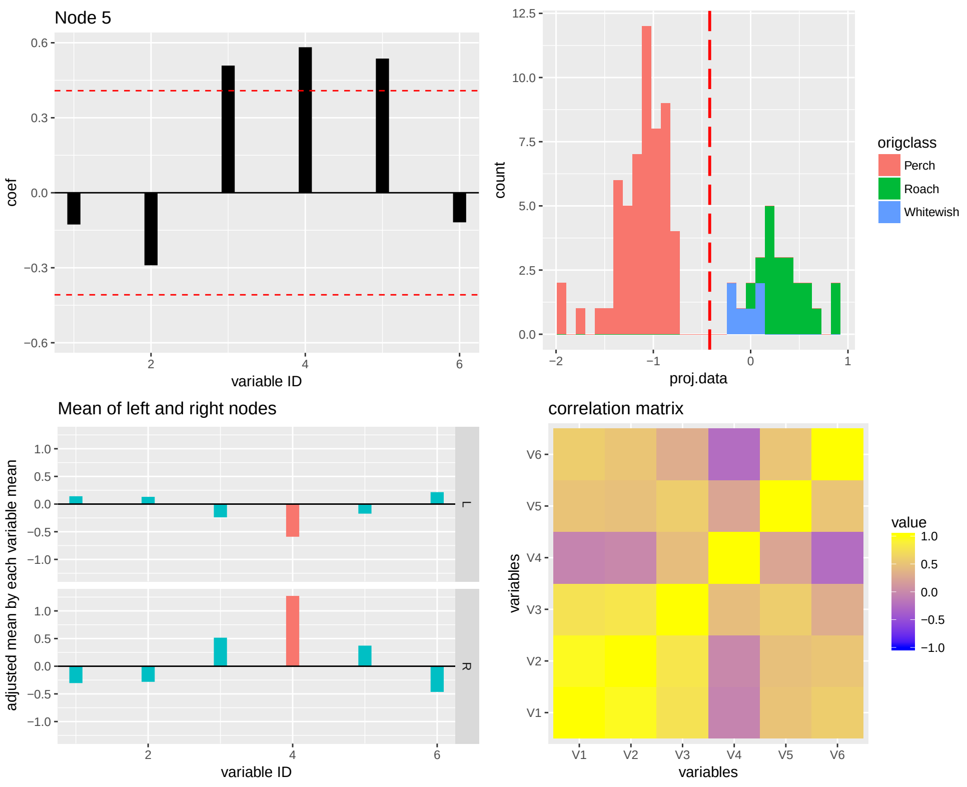


Figure 12: Node 5 of fishery data.

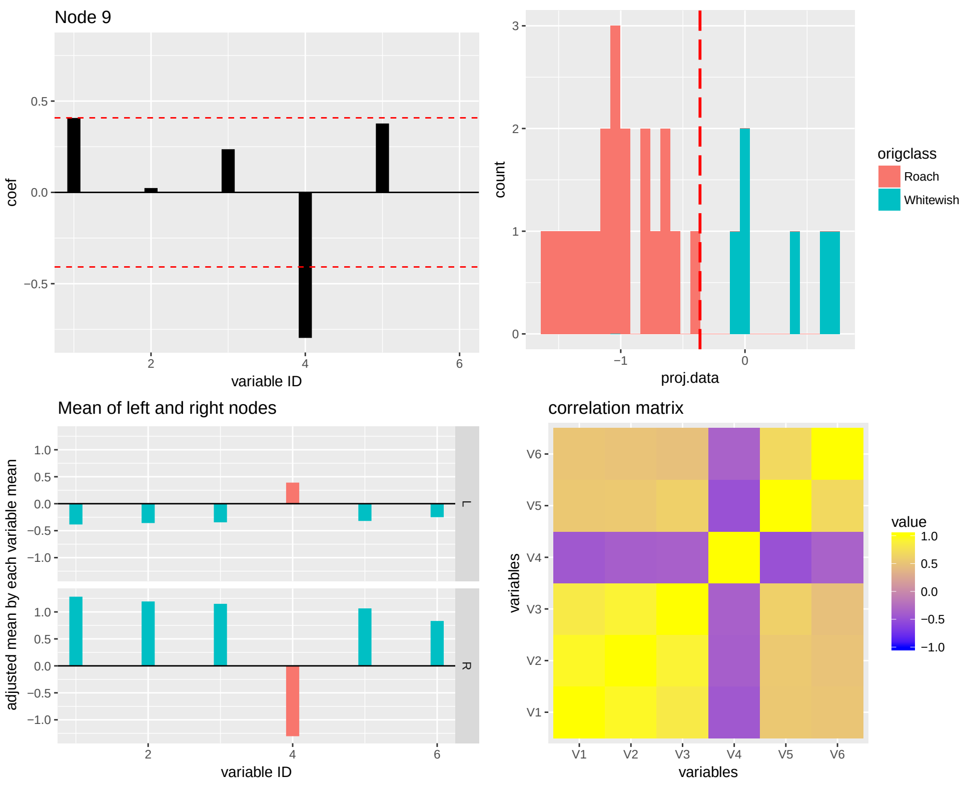


Figure 13: Node 9 of fishery data.

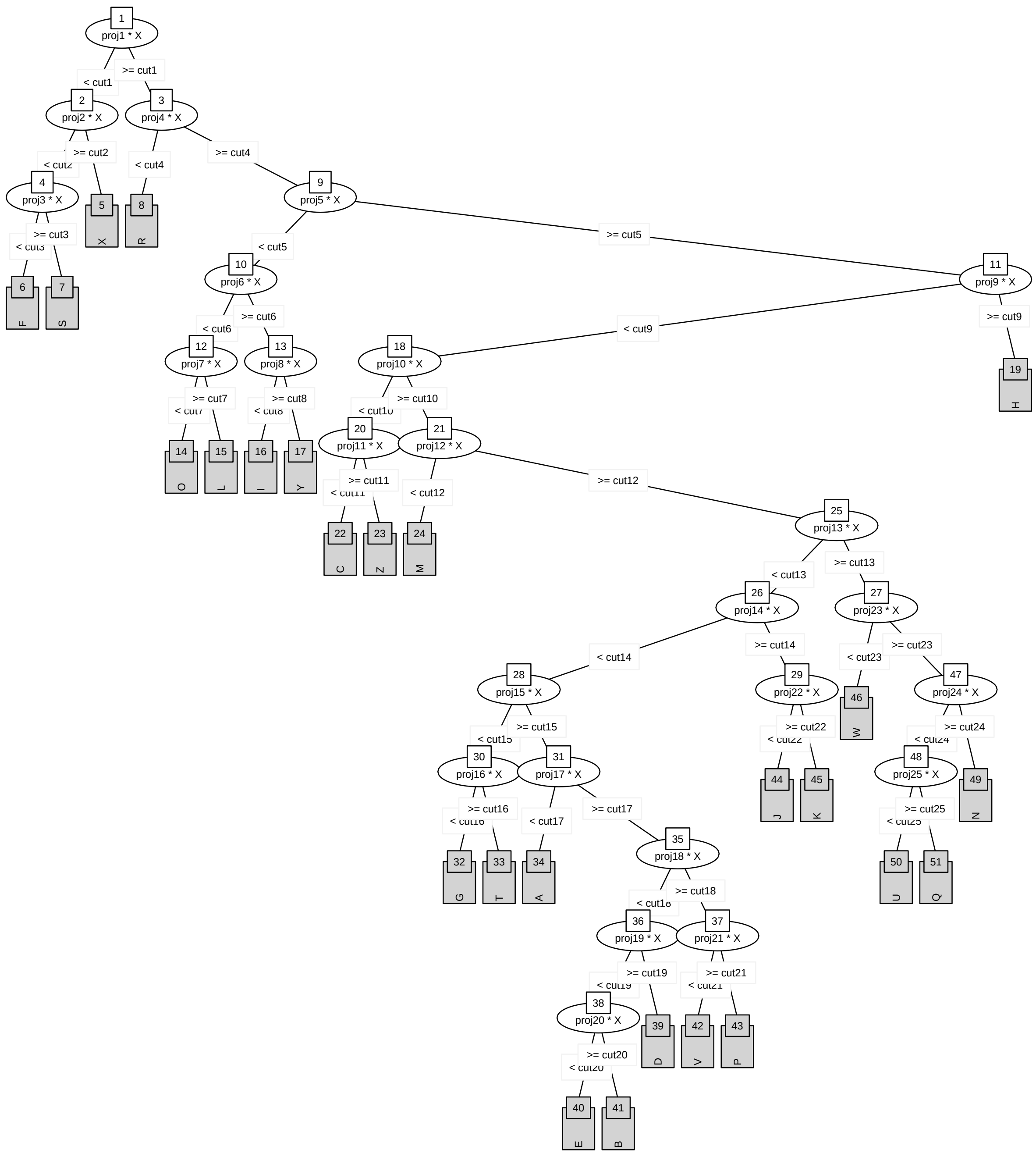


Figure 14: Projection pursuit classification tree with letter identification data.

has 60 observations for each letter. Both data sets have 617 variables. The descriptions of the variables without matching the order in the data sets can be found in Cole and Fanty (1990). This data set was adopted in several studied in machine learning (Cole and Fanty 1990, Dietterich and Bakiri 1991). The algorithms are mainly designed to precisely predict letters with 617 variables from spoken letters. Their concern is focused on predictability, not interpretability. However it is also important to determine the feature of each letter and to

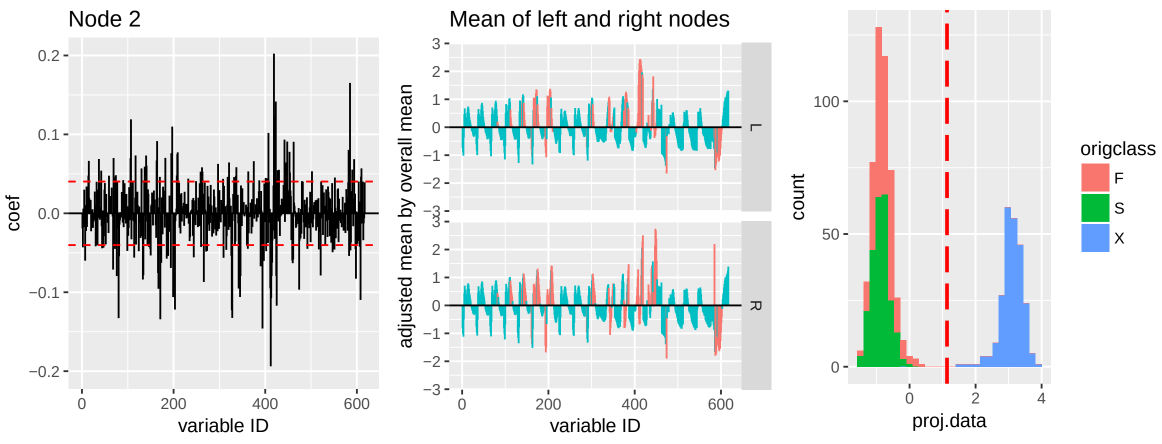


Figure 15: Node 2 of projection pursuit classification tree with letter identification data.

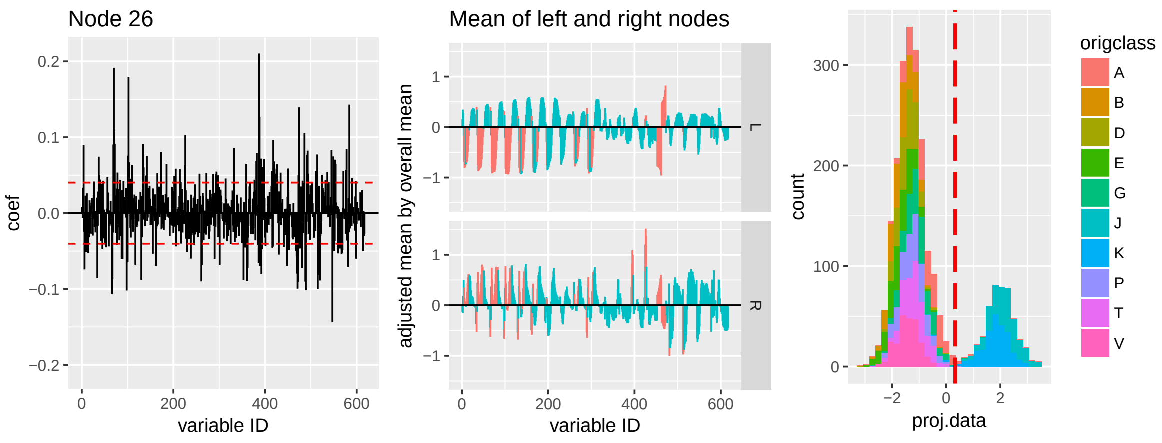


Figure 16: Node 26 of projection pursuit classification tree with letter identification data.

understand their characteristics. In this paper, we have explored the tree structure of the projection pursuit classification tree using the training set and the PDA index with `lambda = 0.1`. Figure 14 shows the entire projection pursuit classification tree structure of the ISOLET data. The projection pursuit classification tree has 26 final nodes and the depth of this tree is 14. With the `plot` function in **PPtreeViz**, we can draw this complicated tree structure without any overlap. The training error rate for this projection pursuit classification tree is 0.0224 and the test error rate is 0.0693.

Figure 15 presents the results for node 2. In this ISOLET data, each variable is represented as a line in the coefficient plot and the mean plot since we have too many variables. In the mean plot, the variables with a large difference between the left and right group are shown in red. Node 2 separates the letter F and S from the letter X. These two groups of letters have quite different patterns, especially around variable 350 and 450. For these variables, the coefficient plot has high peaks. Also the means between variables 150 and 220, and around variable 600 are different. Figure 16 shows the results for node 26, where J and K are separated from G, T, A, E, B, D, V, and P. The main differences of the means between these two groups of letters are in the first 200 variables, the variables between 450 and 470, and around variable 400.

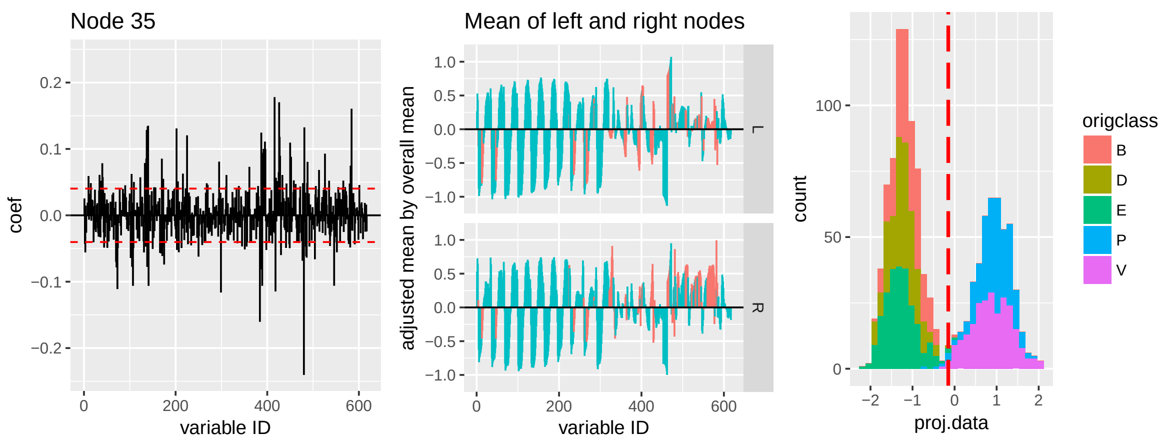


Figure 17: Node 35 of projection pursuit classification tree with letter identification data.

This feature is also captured by the coefficient plot. Figure 17 shows node 35 separating E, B and D from V and P. The patterns between variable 450 and variable 600, around variable 330, and around variable 420 differ somewhat, and the coefficient plot has high peaks around these variables. Due to the lack of information regarding the description of the variables, we cannot clearly explain how these letters are differently classified. However, we determine which variables mostly differ between the two groups and how they separate groups.

```
=============================================================
Projection Pursuit Classification Tree result
=============================================================

1) root
   2)  proj1*X < cut1
      4)  proj2*X < cut2
         6)* proj3*X < cut3  ->  "F"
         7)* proj3*X >= cut3  ->  "S"
      5)* proj2*X >= cut2  ->  "X"
   3)  proj1*X >= cut1
      8)* proj4*X < cut4  ->  "R"
      9)  proj4*X >= cut4
         10)  proj5*X < cut5
            12)  proj6*X < cut6
               14)* proj7*X < cut7  ->  "O"
               15)* proj7*X >= cut7  ->  "L"
            13)  proj6*X >= cut6
               16)* proj8*X < cut8  ->  "I"
               17)* proj8*X >= cut8  ->  "Y"
         11)  proj5*X >= cut5
            18)  proj9*X < cut9
               20)  proj10*X < cut10
                  22)* proj11*X < cut11  ->  "C"
                  23)* proj11*X >= cut11  ->  "Z"
               21)  proj10*X >= cut10
                  24)* proj12*X < cut12  ->  "M"
                  25)  proj12*X >= cut12
                     26)  proj13*X < cut13
                        28)  proj14*X < cut14
                           30)  proj15*X < cut15
                              32)* proj16*X < cut16  ->  "G"
                              33)* proj16*X >= cut16  ->  "T"
                           31)  proj15*X >= cut15
                              34)* proj17*X < cut17  ->  "A"
                              35)  proj17*X >= cut17
                                 36)  proj18*X < cut18
                                    38)  proj19*X < cut19
                                       40)* proj20*X < cut20  ->  "E"
                                       41)* proj20*X >= cut20  ->  "B"
                                    39)* proj19*X >= cut19  ->  "D"
                                 37)  proj18*X >= cut18
                                    42)* proj21*X < cut21  ->  "V"
                                    43)* proj21*X >= cut21  ->  "P"
                        29)  proj14*X >= cut14
                           44)* proj22*X < cut22  ->  "J"
                           45)* proj22*X >= cut22  ->  "K"
                     27)  proj13*X >= cut13
                        46)* proj23*X < cut23  ->  "W"
                        47)  proj23*X >= cut23
                           48)  proj24*X < cut24
                              50)* proj25*X < cut25  ->  "U"
                              51)* proj25*X >= cut25  ->  "Q"
                           49)* proj24*X >= cut24  ->  "N"
            19)* proj9*X >= cut9  ->  "H"

Error rates of various cutoff values
-------------------------------------------------------------
            Rule1  Rule2  Rule3  Rule4  Rule5  Rule6  Rule7  Rule8
error.rate 0.0224 0.1347 0.0459 0.2774 0.0228 0.1449 0.0569 0.3431
```

6. Discussion

The projection pursuit classification tree is a tree for classification that is used with the projection pursuit method. Each class in this tree is assigned to only one final node,which simplifies the final tree structure. The final tree structure is worth exploring, since it is easy to understand, to interpret how classes are separated and to explore the feature of each class. The original R package **PPtree** for the projection pursuit classification tree was developed using the C language for the main calculation. We want to keep **PPtree** with the C language and also to make a new version for calculation with **Rcpp**. Therefore, we developed the projection pursuit index calculation, optimization algorithm with **Rcpp** and **RcppArmadillo** packages. We also add a visualization of the result of the optimization for the projection pursuit indices, the tree structure, and the data space in each node. We combine all these methods into the **PPtreeViz** package.

There are a couple of R packages for the classification tree with a visualization of the tree structure. **rpart** and **rpart.plot** are packages for the classification tree and for visualization of the result of `rpart`. The generic plot of **rpart** does not show good presentation of the tree structure. `prp` provides a more efficient plot of the tree structure including interactive pruning option. `fancyRpartPlot` provides nice presentation of the tree structure. However `fancyRpartPlot` is not suitable for a big tree with a large number of nodes.

The **party** and **partykit** packages provide a function for new classification tree method and also provides a new approach to visualize the the general binary tree. Its visualization method includes options of various plots that present the status of the final node, as well as the tree structure. In its visualization of the tree structure, the inner nodes are represented as ellipses, with an indication of the node, and the final nodes are represented as various type of plots. Even though the visualization methods in **party** and **partykit** are useful, there is a limit to exploring the projection pursuit classification tree.

The visualization of the tree structure in the **PPtreeViz** package adapted the method from the **party** package for the inner node representation. For the final node, **PPtreeViz** presents the assigned class with the node id, and this helps indicate the specific node with the node id and explores the separations in each node as well as the overall tree structure. With this **PPtreeViz**, we can easily understand how classes are separated, which features differ from the other classes, etc. This information can be useful for further analysis in order to improve the predictability of the classification. **PPtreeViz** also provides methods to explore the projection pursuit indices using the class information.

In this paper, we propose two new indices, Gini and entropy, and we provide `GINIindex1D` and `ENTROPYindex1D` functions to calculate the index values. According to the index definition, these indices need to calculate $P_{\text{Gini},k}$ and $P_{\text{Entropy},k}$ for all $k = 1, \ldots, n$, and as well as find the maximum values. It takes more time to get these index values than with other indices (LDA, PDA, or Lp indices), even though we use **Rcpp** (Eddelbuettel and François 2011). It gets worse when we use these indices in the `PPopt` function or `PPtreeClass` function. We recommend not to use `PPmethod = "GINI"` or `PPmethod = "ENTROPY"` options in `PPtreeClass` with a large number of observations or variables.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2009-0093827).

References

Auguie B, Antonov A (2015). ***gridExtra**: Miscellaneous Functions for **grid** Graphics.* R package version 2.3, URL `https://CRAN.R-project.org/package=gridExtra`.

Cole R, Fanty M (1990). "Spoken Letter Recognition." In *Proceedings of the Third DARPA Speech and Natural Language Workshop*, pp. 385–390.

Dietterich TG, Bakiri G (1991). "Error-Correcting Output Codes: A General Method for Improving Multiclass Inductive Learning Programs." In *AAAI*, pp. 572–577. Citeseer.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.

Hastie T, Friedman J, Tibshirani R (2011). *The Elements of Statistical Learning*. Springer-Verlag, New York. doi:10.1007/978-0-387-21606-5.

Hothorn T, Hornik K, Strobl C, Zeileis A (2017). ***party**: A Laboratory for Recursive Partytioning.* R package version 1.2-4, URL https://CRAN.R-project.org/package=party.

Hothorn T, Zeileis A (2015). "**partykit**: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**, 3905–3909. URL http://jmlr.org/papers/v16/hothorn15a.html.

Huber PJ (1985). "Projection Pursuit." *The Annals of Statistics*, **13**(2), 435–475. doi:10.1214/aos/1176349519.

Huber PJ (1990). "Data Analysis and Projection Pursuit." *Technical Report PJH-90-1*, MIT.

Johnson RA, Wichern DW (2007). *Applied Multivariate Statistical Analysis*. 6 edition. Pearson Prentice Hall, Upper Saddle River.

Lee EK, Cook D (2010). "A Projection Pursuit Index for Large p Small n Data." *Statistics and Computing*, **20**(3), 381–392. doi:10.1007/s11222-009-9131-1.

Lee EK, Cook D, Klinke S, Lumley T (2005). "Projection Pursuit for Exploratory Supervised Classification." *Journal of Computational and Graphical Statistics*, **14**(4), 831–846. doi:10.1198/106186005x77702.

Lee YD, Cook D, Park JW, Lee EK (2013). "**PPtree**: Projection Pursuit Classification Tree." *Electronic Journal of Statistics*, **7**, 1369–1386. doi:10.1214/13-ejs810.

Milborrow S (2017). ***rpart.plot**: Plot `rpart` Models: An Enhanced Version of `plot.rpart`.* R package version 2.1.2, URL https://CRAN.R-project.org/package=rpart.plot.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. Vienna, Austria. R Foundation for Statistical Computing, URL https://www.R-project.org/.

Ripley BD (2017). ***MASS**: Support Functions and Datasets for Venables and Ripley's MASS.* R package version 7.3-48, URL https://CRAN.R-project.org/package=MASS.

Therneau T, Atkinson B, Ripley B (2017). ***rpart**: Recursive Partitioning and Regression Trees.* R package version 4.1-11, URL https://CRAN.R-project.org/package=rpart.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4/.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York. URL http://ggplot2.org/.

Affiliation:

Eun-Kyung Lee
Department of Statistics
Ewha Womans University
52, Ewhayeodae-gil, Seodaemun-gu, Seoul, 120-750, Korea
E-mail: lee.eunk@ewha.ac.kr

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
February 2018, Volume 83, Issue 8 *Submitted:* 2015-09-25
doi:10.18637/jss.v083.i08 *Accepted:* 2017-04-14