Tutorial: Getting Started with MART in R

Jerome H. Friedman
Stanford University

May 13, 2002

Abstract

Multiple additive regression trees (MART) is a methodology for predictive data mining (regression and classification). This note illustrates the use of the R/MART interface. It is intended to be a tutorial introduction. Minimal knowledge concerning the technical details of the MART methodology or the use of the R statistical package is presumed.

1 Introduction

Predictive data mining is concerned with constructing statistical models from historical data. These models are used to predict future unknown data values, and/or to help gain an understanding of the predictive relationships represented in the data. The data consists of sets of measurements made on objects (observations) collected in a data base. One of the measured variables (denoted by y) is designated as the one to be predicted, given future values of the other variables denoted by $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$. Depending on the field of study, y is referred to as the *response* variable (Statistics), *output* variable (neural nets), or *concept* (Machine Learning). The $\mathbf{x}$–variables are referred to as *predictor* variables, *input* variables, or *attributes* in these respective fields.

The data base consists of a collection of N previously solved cases

$$\{y_i, x_{i1}, \cdots, x_{in}\}_{i=1}^{N}. \tag{1}$$

The predictive model takes the abstract form

$$\hat{y} = \hat{f}(x_1, \cdots, x_n), \tag{2}$$

where $\hat{f}$ is a prediction rule that maps a set of predictor variable values to a response value. The goal is to use the data to produce an accurate mapping. The notion of accuracy depends on the type of the response variable y in terms of the set of values it can assume. If y assumes numeric values the problem is known as *regression* and lack of accuracy is defined in terms of a distance measure $d(y, \hat{y})$ between the predicted value $\hat{y}$ and the unknown true value y. Common measures of inaccuracy are average absolute error

$$AAE = ave\,|y - \hat{y}| \tag{3}$$

or root mean squared error

$$RSME = \sqrt{ave\,(y - \hat{y})^2}.$$

Here *ave* represents the average over future values to be predicted. If y assumes unorderable categorical values (class labels), the problem is called *classification.* In this case inaccuracy is generally defined to be the fraction of incorrect future predictions (error rate).

Multiple additive regression trees (MART) is a particular methodology (among many) for trying to solve prediction problems. It is described in detail in Friedman (1999a, 1999b). Besides accuracy, its primary goal is robustness. It tends to be resistant against moderate to heavy contamination by bad measurements (outliers) of the predictors and/or the responses, missing values, and to the inclusion of potentially large numbers of irrelevant predictor variables that have little or no effect on the response .

This note describes the use of MART with the R statistical package. It is presumed that both R and the R/MART interface have been installed. The R statistical package can be obtained from

http://www.r-project.org/.

The R/MART interface can be obtained from

$$\text{http://www-stat.stanford.edu/\~{}jhf/R-MART.html.} \tag{4}$$

This note is intended for beginning users. Only the most basic uses of the MART package are described. Most options and "tuning" parameters are set to defaults. As users become familiar with the basic procedures, they can experiment with more advanced options. All of the options associated with each MART procedure in R are described in detail in their associated documentation (help files) included with the installation.

The R/MART interface consists of commands entered in the R command window. These commands return either numeric results or plots. It is assumed that the user is familiar with how to invoke R and enter commands in its command window.

2 Reading in data

The primary input to MART is the historical data (1) used to construct the predictive model (2). This data must be read into R. R is able to input data from a wide variety of data producing applications. (See R documentation.) The most straightforward way is to store the data in some directory as ASCII (text) file(s). They can then be read into R using the *scan* command. Suppose the predictor variable values are stored in the file *datadir*/xdata.txt where *datadir* is the full path name of a selected directory. Entering

```
> x_scan('datadir/xdata.txt')
```

at the R command prompt ">" will store this data under the name "x" in R. Similarly, if the response values were stored in the file ydata.txt in the same directory, the command

```
> y_scan('datadir/ydata.txt')
```

stores them under the name "y" in R. The actual data files (xdata and ydata) used for the illustrations below can be downloaded from (4).

The R *mart* procedure requires that the predictor variable values be in the form of a matrix whose rows represent observations and columns represent predictor variables. The values in "x" can be converted to such a matrix with the R *matrix* command

```
> x_matrix(x, ncol = nvar, byrow=T)
```

where *nvar* is the number of predictor variables. The argument "byrow=T" indicates that the predictor variable values were stored in the file xdata.txt in observation sequence. That is, the predictor variable values for each observation follow those of the previous observation. If the data were stored in variable sequence, where all of the observation values for a predictor variable follow those of the previous variable, then the last argument would be omitted. For the illustrative data used below, there are ten predictor variables stored in observation sequence, so the command would be

```
> x_matrix(x, ncol = 10, byrow=T)
```

The observations now correspond to the rows of the matrix "x". For the illustrative data set the command

```
> nrow(x)
```

```
[1] 10000
```

shows that this data set consist of 10000 observations.

The *mart* procedure must be told the type of variable (numeric or categorical) of each predictor variable. This information can be stored in a vector of flags "lx" of length *nvar*. Each element of "lx" corresponds to a predictor variable

$$\mathrm{lx}\,[\,\mathrm{j}\,] = \begin{cases} 0 & \text{ignore } x_j \\ 1 & x_j \text{ is numeric} \\ 2 & x_j \text{ is categorical} \end{cases} .$$

Binary variables can be considered either numeric or categorical. Numeric specification provides a speed advantage. For the illustrative data set there are two categorical variables (x_3 and x_4) and the rest are numeric. This would be in indicated with the R command

```
> lx_c(1,1,2,2,rep(1,6))
```

Here "c(·)" is the R "concatenate" operator, and the procedure *rep(i,j)* simply repeats the value *i*, *j* times. An alternative would be

```
> lx_c(1,1,2,2,1,1,1,1,1,1)
```

The *mart* default is that all predictor variables are numeric. So if this is the case, the "lx" parameter need not be specified in the *mart* command.

Finally, the *mart* procedure must be told which entries in the predictor data matrix have missing values. Each missing value must be represented by a numeric missing value flag *missval*. The value of this flag must be larger than any nonmissing data value. The *mart* default is *missval* = 9.0e30. If this default value is used, or if there are no missing values and no data value is greater than 9.0e30, then the parameter "xmiss" need not be specified as an argument to the *mart* command. Otherwise, the actual value of *missval* that was used must be specified by setting xmiss = *missval* in the argument list of the *mart* command. For the illustrative data set there are no missing values.

3 Regression

For regression the MART model is constructed using the *mart* command

```
> mart(x,y,lx)
```

This command starts a task independent of R running in a separate window. The real time progress of model construction can be viewed in this window. MART is an iterative procedure. Each iteration increases the complexity (flexibility) of the model. Shown in the MART window are the current number of iterations, and the corresponding "training" and "test" set average absolute error (3) of the MART model at that iteration. Internally, *mart* randomly partitions the input data into a "training" data set and a complement "test" set. Only the training set is used to construct the model; the test set is used to provide an unbiased error estimate. The default size of the test set is (1/5)th that of the total input data set.

The default number of iterations is 200. When completed the MART window automatically closes. A numerical summary of the MART run is then displayed in the command window:

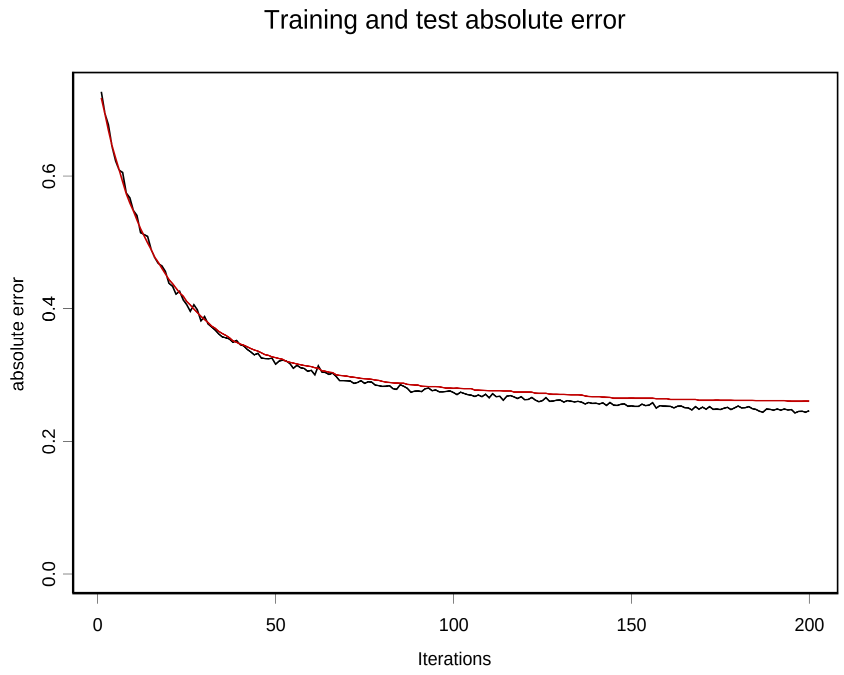

Figure 1: Output from the command `progress()` after 200 iterations.

```
MART execution finished.
 iters     best     test abs(error)
  200       197         0.2607
```

This result indicates that there have been 200 total iterations so far, the smallest test error was obtained at iteration 197 with a corresponding average absolute error of 0.2607.

A graphical summary can be obtained by issuing the command

```
> progress()
```

This produces the plot shown in Fig. 1. The lower (black) line is the training error as a function of number of iterations. The upper (red) line is the corresponding test error.

From Fig. 1 it appears that there is still a general downward trend in test error when the iterations finished. This suggests that more iterations may be beneficial. Iterations can be continued from where the last run ended by the command

```
> moremart()
```

This again creates a separate MART task that performs 200 (default) more iterations. After this task completes, a summary of the current model is presented:

```
MART execution finished.
 iters     best     test abs(error)
  400       400         0.2534
```

This shows that the additional iterations did improve test absolute error, if only by about 3%. Entering

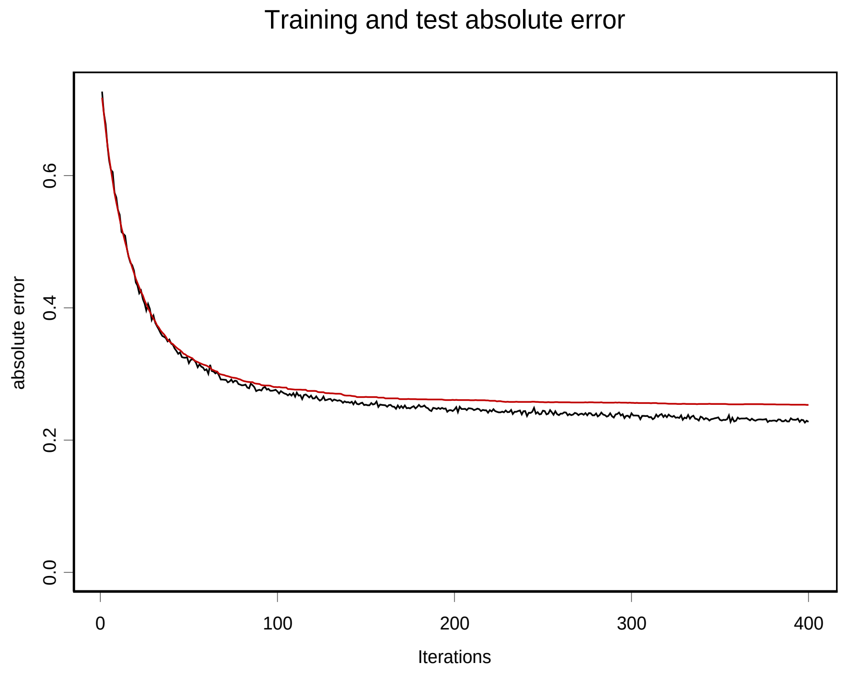

Figure 2: Output from the command `progress()`after 400 iterations.

```
> progress()
```

now produces the plot shown in Fig. 2.

Prediction error is still getting smaller at the 400th iteration. This can be more clearly seen with the command

```
> progress(F)
```

which shows a magnified version (Fig. 3) of just the iterations corresponding to the last *moremart* command. The noisy appearance of the training error is due to the stochastic sampling described in Friedman 1999b. One could continue iterating in this manner by issuing further *moremart* commands. However, it might be useful to pause and examine the predictive model at this point. More iterations can be applied at any time in the future using additional *moremart* commands.

3.1 Interpretation

The command

```
> varimp()
```

produces the plot shown in Fig. 4. This shows the estimated relative importance (relevance or influence) of each predictor variable in predicting the response based on the current model. The variables are ordered in decreasing importance. Here one sees that variables $x_1 - x_4$ are all highly and nearly equally important. Variable x_5 is somewhat less important. Variables $x_6 - x_{10}$ are seen to have little relevance for prediction.

After identifying the most relevant predictor variables, the next step is to get an idea of the dependence of the MART model on each of them. This is done through "partial dependence" plots. Entering the command

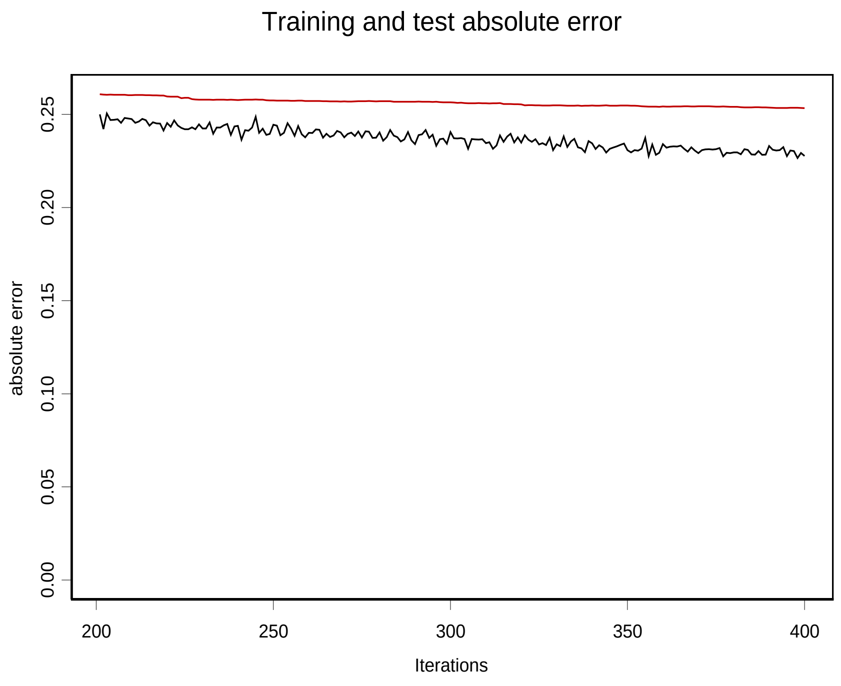

Figure 3: Output from the command `progress(F)` after 400 iterations.

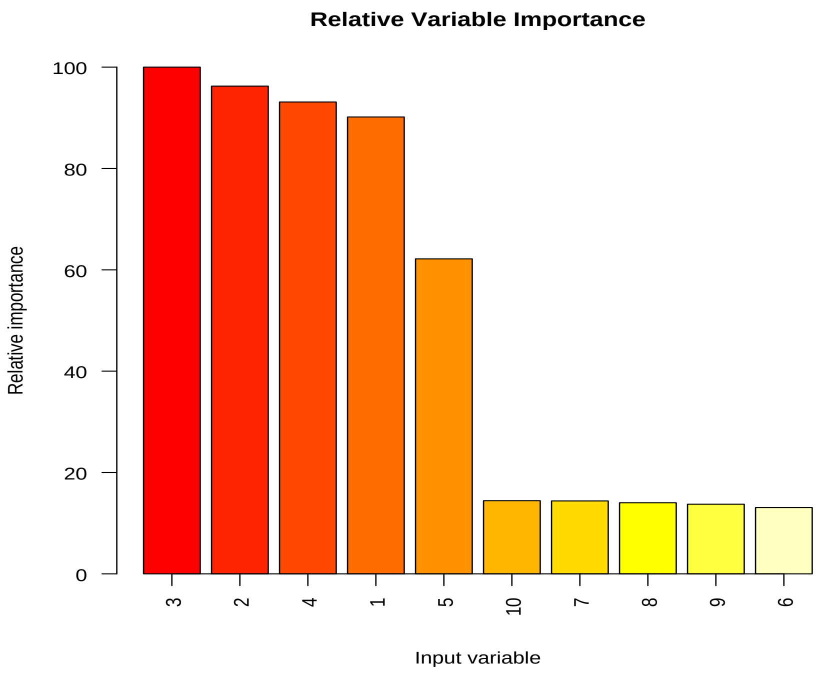

Figure 4: Output from the command `varimp()`.

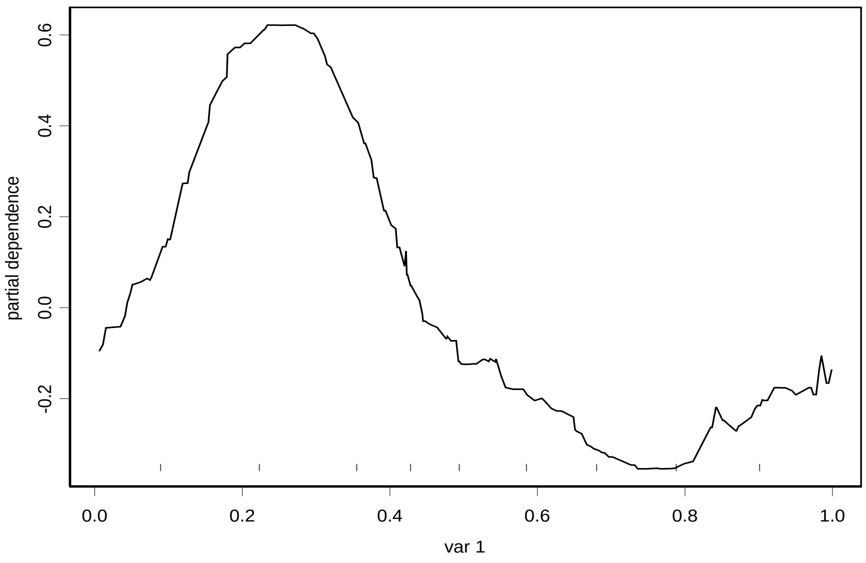

Figure 5: Output from the command `singleplot(1)`.

```
> singleplot(1)
```

produces the plot shown in Fig. 5. This is the partial dependence of the model (2) on predictor variable x_1, after accounting for the (average) joint effect of the other predictors $(x_2, \cdots, x_{10})$. The hash marks at the base of the plot represent the deciles of the x_1 distribution. In general, the nature of the dependence of the model on this variable will depend on the values of the other predictor variables, and cannot be represented on a single plot. The partial dependence plot shows the dependence on x_1 as averaged over the distribution of values of the other variables $\{x_2, x_3, \cdots, x_{10}\}$. While this may not provide a comprehensive description, it can show general trends.

Partial dependence plots for the other predictor variables are obtained using the *singleplot* command with the corresponding variable number in the argument. Figure 6 collects together the results for the first six predictor variables. All plots are centered to have zero mean. Partial dependence plots for categorical variables (here x_3 and x_4) are displayed as horizontal barplots, each bar representing one of the categorical values. The values are (bottom to top) ordered on increasing value when the category label is interpreted as a real number. (For this example the labels for x_3 were encoded as 0,1,2,3, and those for x_4 are 0,1,2.)

Note that none of the functions of the numeric predictors are strictly smooth. Unlike most other modeling procedures, MART imposes no explicit smoothness constraint on the solution. It is able to model arbitrarily sharp changes in the response. The extent to which the plots exhibit smooth general trends is dictated by the data. Here a smoother model tends to fit better. Also note the much smaller vertical scale on the plot for x_6 (lower right). This dependence looks like pure noise and seems to have no general trend. The plots for predictor variables x_7 - x_{10} look similar. This is consistent with Fig. 4 that indicated that $x_6 - x_{10}$ have little to no predictive power.

The MART procedure *singleplot* plots partial dependence on individual predictor variables. It is also possible to plot partial dependences on pairs of predictor variables. The command

```
> pairplot(1,2)
```

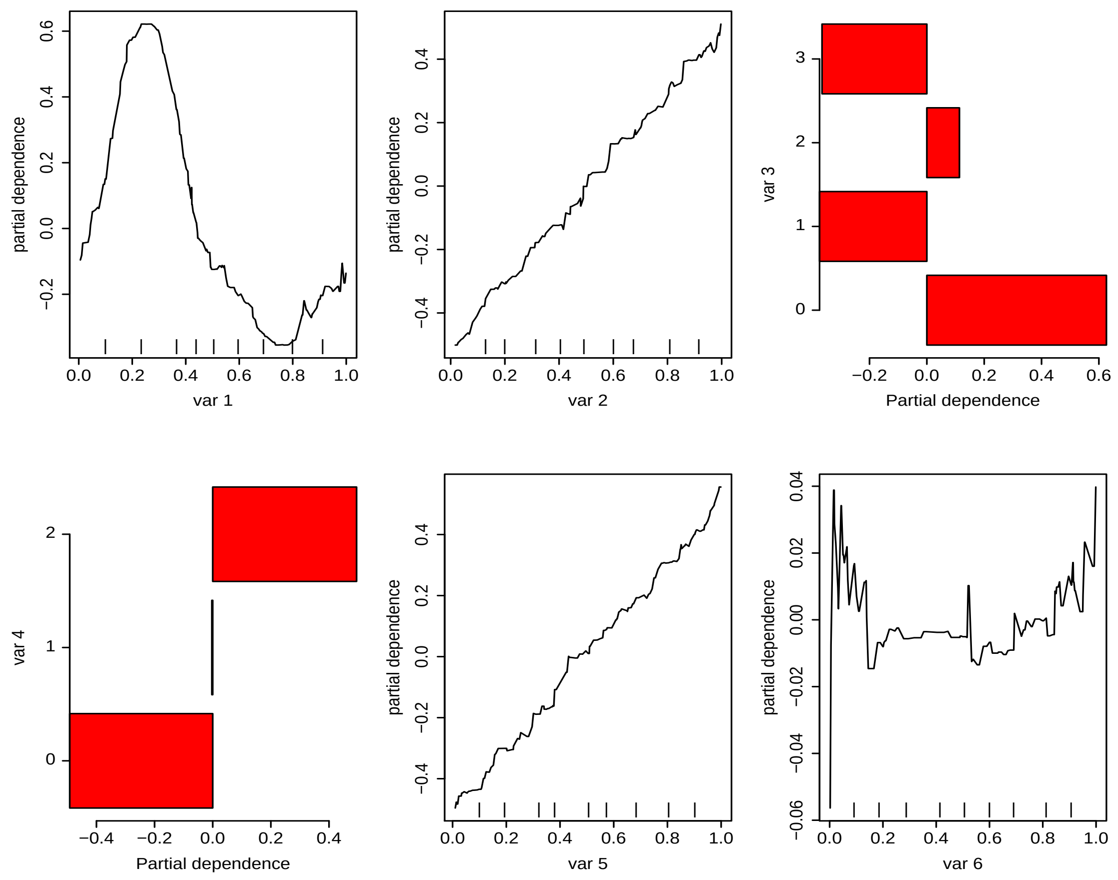

Figure 6: Output of the commands `singleplot(1)`; · · ·; `singleplot(6)`.

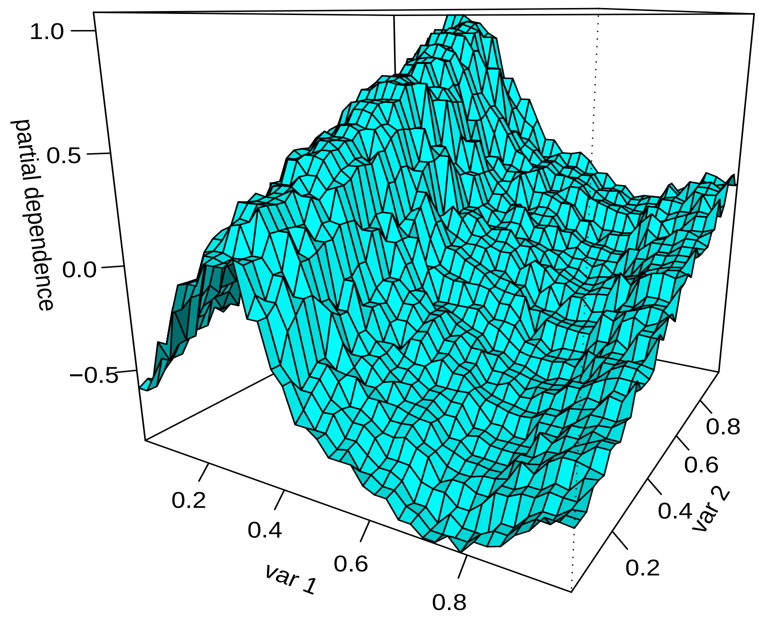

Figure 7: Output of the command `pairplot(1,2)`.

produces the picture shown in Fig. 7. This is a perspective mesh plot showing the dependence of the predictive model (2) on joint values of (x_1, x_2) after accounting for the average joint effect of the other eight predictor variables. This joint dependence is seen to be fairly additive. The shape of the dependence on either variable is unaffected by the value of the other variable. This suggests that there is no interaction between these two variables.

Entering the command

```
> pairplot(1,3)
```

produces a series of plots as shown in Fig. 8. This is because x_3 is a categorical variable. Each successive plot shows the partial dependence of the predictive model on the numeric variable x_1, for each of the respective categorical values of x_3. Here one sees a very strong interaction effect between these two variables. For $x_3 = 0$, the model has little or no dependence on x_1. For other x_3–values, the shape of the dependence changes considerably. Figure 8 can be contrasted with Fig. 5. Figure 8 is a more detailed description showing the partial dependence on x_1, for each value of x_3. Figure 5 shows the partial dependence on x_1 as averaged over the values of x_3 in the data base.

Proceeding further, the command

```
> pairplot(1,4)
```

shows (Fig. 9) the partial dependence of the model on x_1 for the three categorical values of x_4. All of the plots reveal the same general dependence indicating little or no interaction between x_1 and x_4.

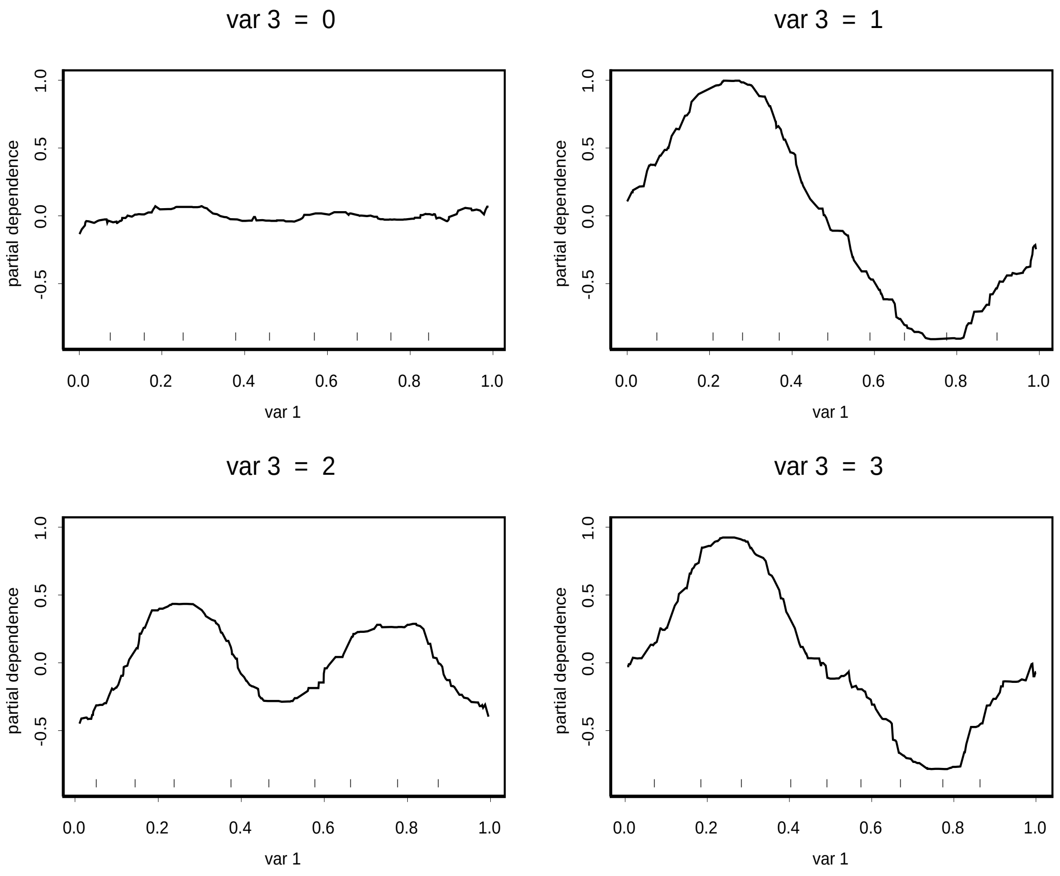

Figure 8: Output of the command `pairplot(1,3)`.

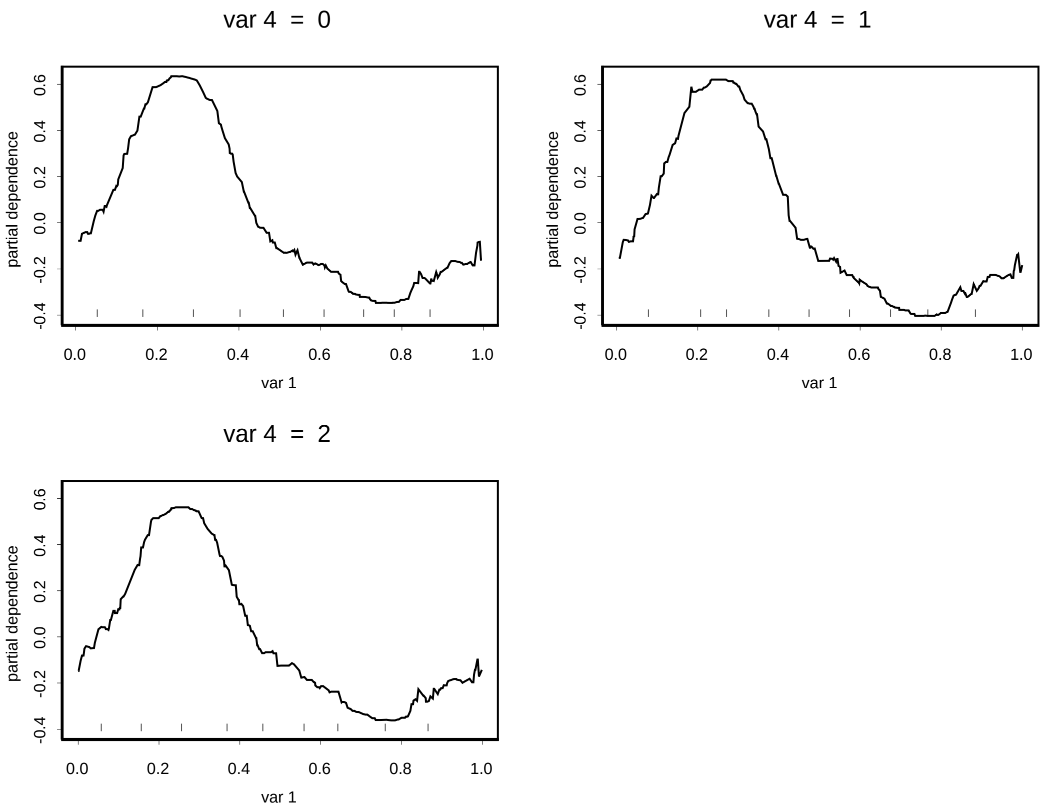

Figure 9: Output of the command `pairplot(1,4)`.

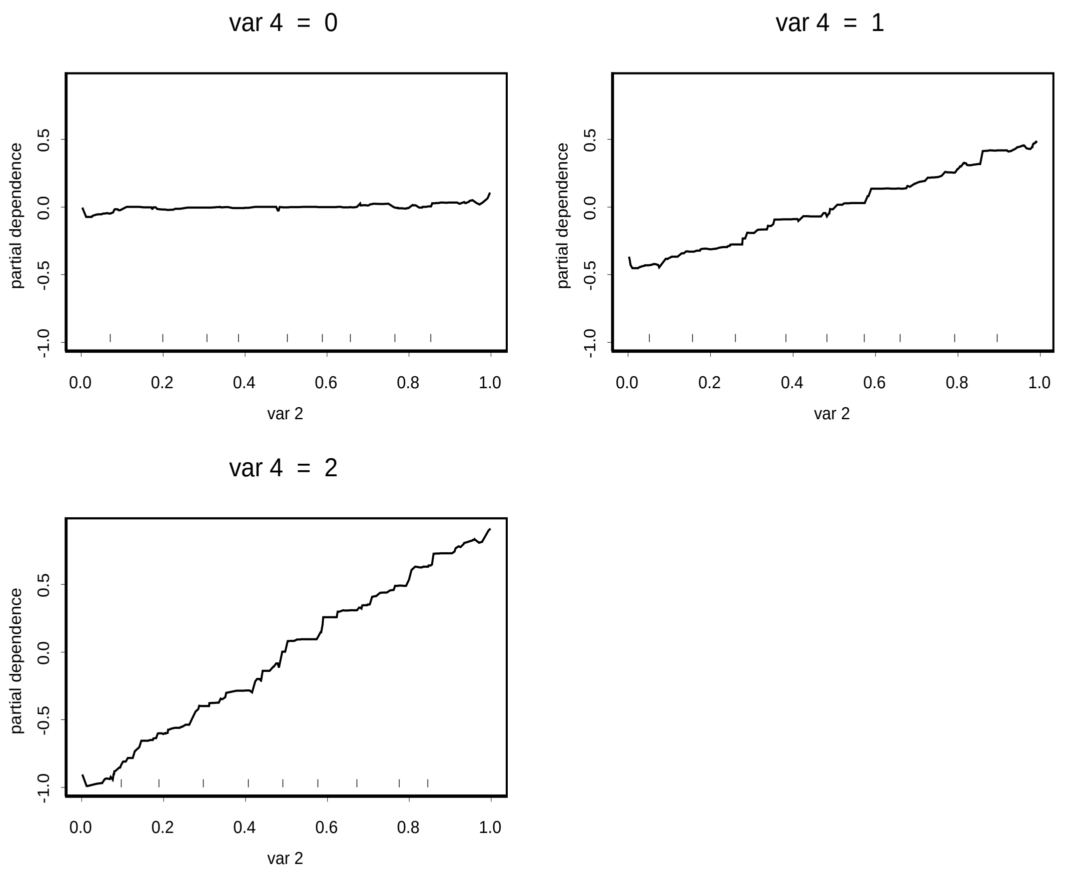

Figure 10: Output of the command `pairplot(2,4)`.

The command

```
> pairplot(2,4)
```

produces the plots shown in Fig. 10. The dependence of the model on x_2 is seen to be generally linear for all values of x_4, but the slope is seen to change for different x_4–values. This suggests a product dependence $f(x_4) \cdot x_2$ on these two predictor variables.

The command

```
> pairplot(3,4)
```

produces a series of barplots (Fig. 11) because both x_3 and x_4 are categorical. Each barplot shows the partial dependence on x_3, conditioned on each of the x_4 values. By default, *pairplot* chooses the variable with the smaller number of values as the one to be conditioned on. One can override this default by specifying the conditioning variable. The command

```
> pairplot(3,4,cvar=3)
```

produces the corresponding plot shown in Fig. 12. Here neither set of plots show much evidence for interactions between these two variables.

By repeatedly applying the procedures *singleplot* and *pairplot*, one can graphically examine the predictive relationship between the predictor variables and the response, as reflected by the

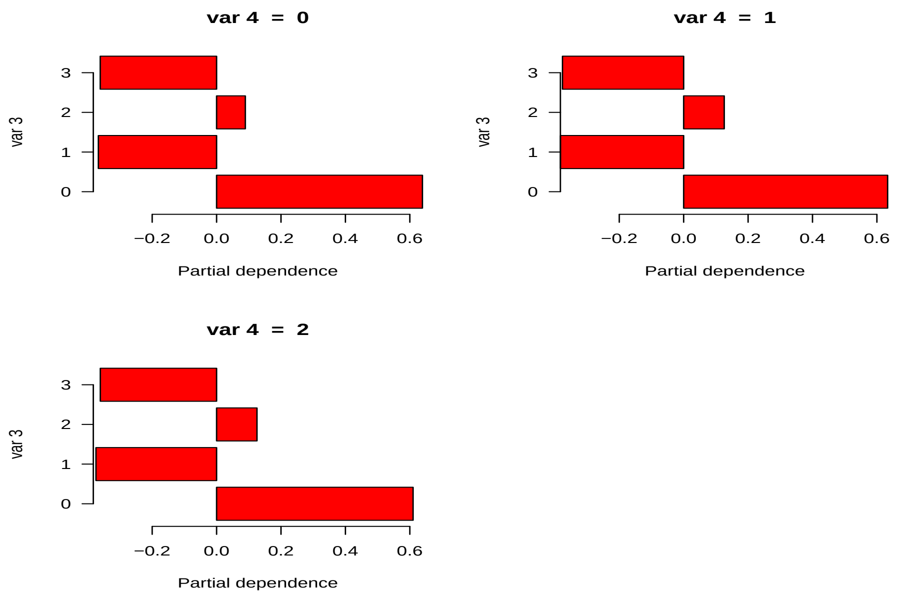

Figure 11: Output of the command `pairplot(3,4)`.

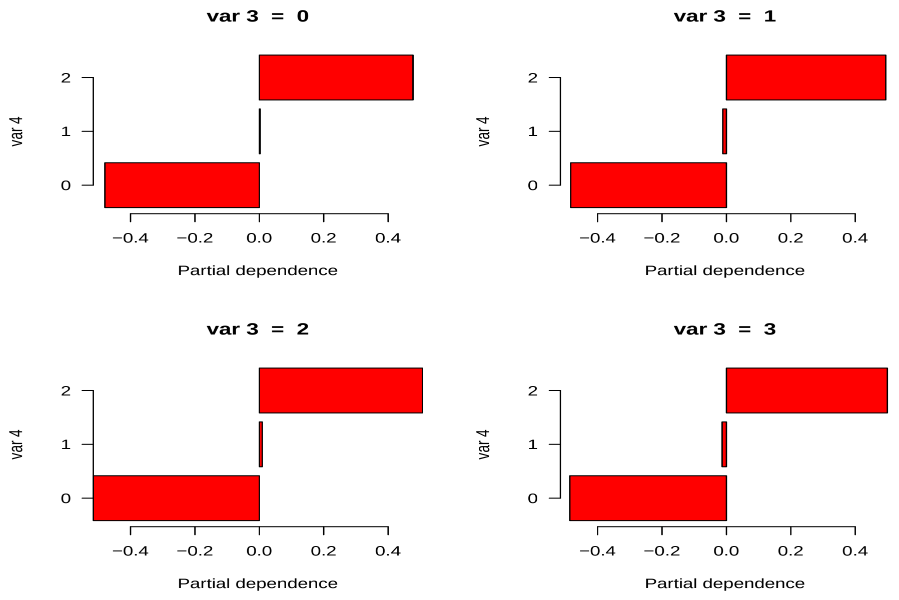

Figure 12: Output of the command `pairplot(3,4,cvar=3)`.

predictive model. Depending on the complexity of this relationship, considerable understanding can often be obtained concerning the system from which the data were collected.

The data used here for illustration were artificially generated according to the prescription

$$y = f(\mathbf{x}) + \sigma \cdot \varepsilon.$$

The eight numeric predictor variables were randomly sampled from a joint uniform distribution $U^8[0,1]$. The two categorical variables were randomly sampled with equal probability assigned to each of their values. The errors were standard normal $\varepsilon \backsim N[0,1]$. The scale of the noise was taken to be $\sigma = \sqrt{var_{\mathbf{x}}[f]}\Big/3$, providing a 3/1 signal-to-noise ratio. The "target" function was

$$f(\mathbf{x}) = [\sin(2\pi x_1)]^{x_3} + x_2 \cdot x_4 + x_5. \tag{5}$$

3.2 Prediction

The MART model can be used to predict (estimate) response values $\hat{y}$ given sets of predictor variable values $\{x_1, x_{2,} \cdot \cdot\cdot, x_n\}$ (2). This is done with the *martpred* command

```
> yh_martpred(xp)
```

Here "xp" is either a vector of length *nvar* representing a single observation, or an *nobs* by *nvar* matrix representing *nobs* observations to be predicted. Here *nvar* is the number of predictor variables (columns of "x"). In the former case a single number is returned to "yh". In the latter case a vector of predictions of length *nobs* is returned. As an example,

```
> xp_c(0.5,0.5,1,2,rep(0.5,6))

> martpred(xp)

[1] 1.474161
```

returns the prediction 1.474161 for the indicated vector "xp".

A matrix of observations to be predicted "xp" can be input to R from an ASCII file in the same manner as was the original data matrix "x"

```
> xp_matrix(scan('datadir/xpred'), ncol = nvar, byrow=T)
```

The resulting vector "yh" can be written to an ASCII file with the R *write* command

```
> write(yh,file='datadir/yhat')
```

4 Classification

For classification problems the response variable y is a label identifying class membership of each observation. The goal of the model is to produce predictions $\hat{y}$ that minimize the misclassification *risk* R (average loss) of future predictions

$$R = ave\, L(\hat{y}, y). \tag{6}$$

Here $L(\hat{y}, y)$ is the loss incurred in predicting $\hat{y}$ when the unknown true label is y. The loss matrix is specified by the user. The default loss matrix in MART is

$$L(\hat{y}, y) = 1(\hat{y} \neq y) \tag{7}$$

so that the risk (6) is the fraction of incorrect predictions $\hat{y} \neq y$. General loss structures can be specified: see *mart* documentation (help file).

For MART in R the class labels can be of any type (numeric or character) and can assume any set of values, provided that observations corresponding to the same class are represented by the same label value. The observation labels can be read into R from an ASCII file by the command

```
> y_scan('datadir/classlab.txt')
```

where *datadir* is the directory where the file is stored. For the illustrations below, the predictor variable data matrix “x” is the same as that used above to illustrate regression. The corresponding class labels (classlab.txt) can be downloaded from (4). The R *scan* command assumes by default that the quantities to be read from the file are numeric values. This is the case for the classlab.txt file (4) used here. If the class labels stored in the file happened to be character values, then the corresponding command

```
> y_scan('datadir/classlab.txt',what=' ')
```

should be used.

The classification MART model is constructed using the *mart* command

```
> mart(x,y,lx,martmode='class')
```

This launches an independent task to build the model. One can monitor construction in real time in the corresponding window. Shown in the window are the current number of iterations, the corresponding test set misclassification risk (6) (7), and the fraction of training observations currently being used to improve the model. MART uses only those observations that are close to the current estimated decision boundary. When 200 iterations (default) are completed, the window automatically closes and the numerical summary is presented:

```
MART execution finished.
 iters     best     test misclass risk
  200       195         0.1835
```

A graphical summary is obtained by

```
> progress()
```

which produces the two plots shown in Fig. 13. The left plot shows test misclassification risk as a function of number of iterations and the right one shows the fraction of training observations being used. Note that the largest fraction is 0.5. This is due to the stochastic sampling being employed (Friedman 1999b).

As seen in the left plot, misclassification risk (here error rate) is still generally decreasing at the end of the previous run. Entering the command

```
> moremart()
```

performs 200 more iterations (default) starting from the previous solution. When completed, the summary

```
MART execution finished.
 iters     best     test misclass risk
  400       370         0.1555
```

shows that the additional iterations have improved the error rate. Again entering

```
> progress()
```

shows (Fig. 14) a graphical summary of the 400 iterations so far. The command

```
> progress(F)
```

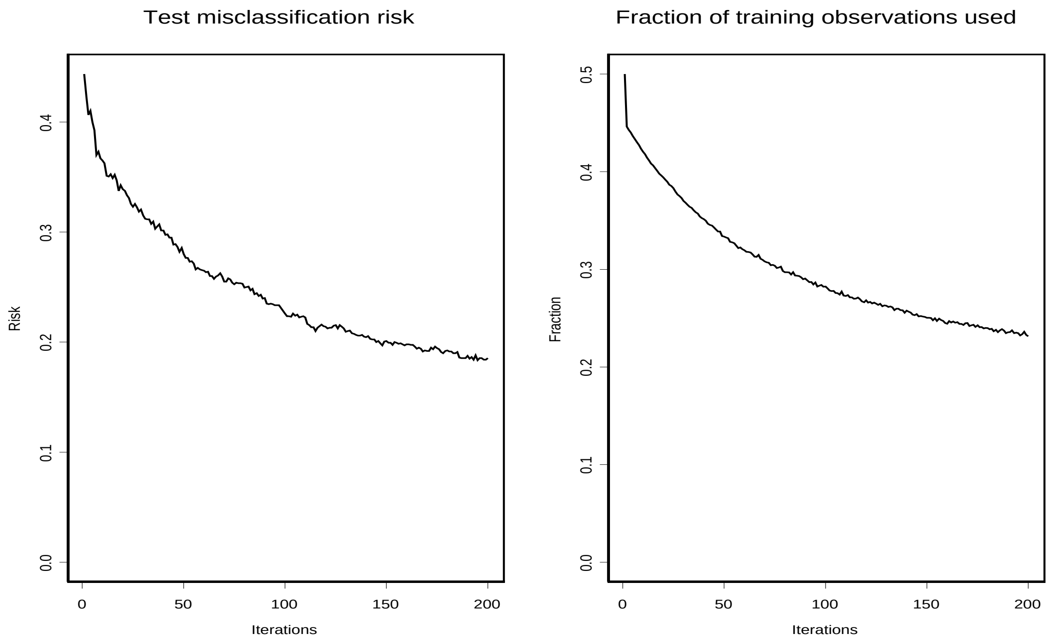

Figure 13: Output of the command `progress()` for classification after 200 iterations.

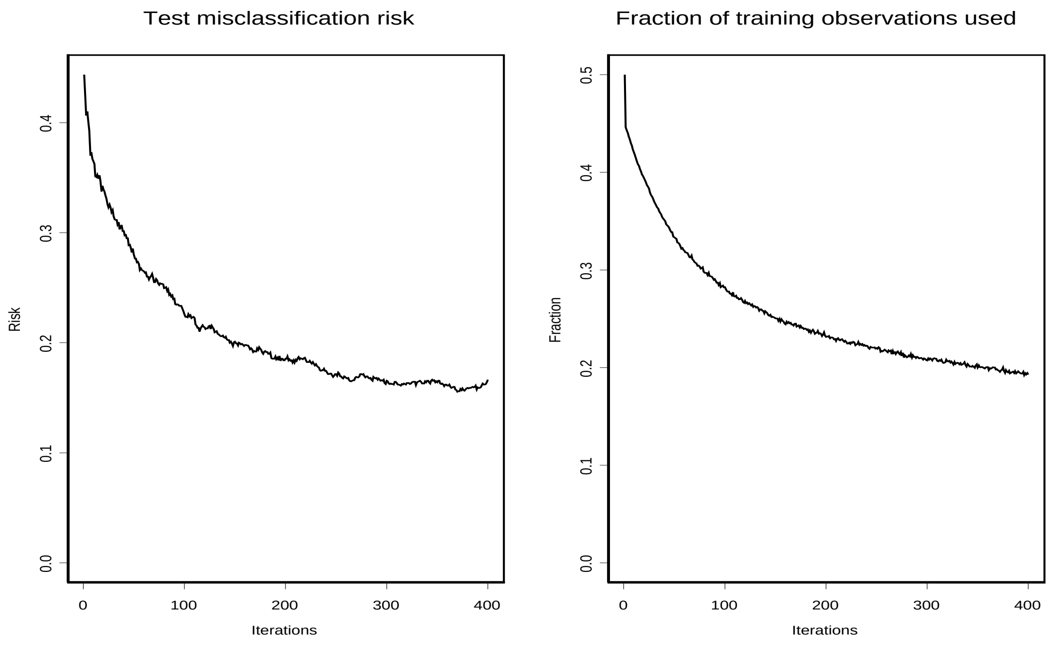

Figure 14: Output of the command `progress()` for classification after 400 iterations.

produces a magnified version for just the iterations corresponding to the last *moremart* command (not shown).

It appears from Fig. 14 that the error rate is on an increasing trend at the 400th iteration. However, it sometimes happens that a local increasing trend can be followed by a later decreasing one. This can be verified by again entering

```
> moremart()
```

to produce 200 more iterations. The corresponding summary

```
MART execution finished.
 iters     best     test misclass risk
  600       556         0.1540
```

shows that a later downward trend did indeed occur, but the final result is estimated to be only slightly better.

4.1 Interpretation

A first step in interpreting classification models is to understand the nature of the misclassifications. Entering the command

```
> classerrors()
```

produces the horizontal barplot shown in Fig. 15. Each bar corresponds to one of the class labels. The labels are shown on the left. For this example we see that there are five classes, each labeled by an integer $k \in \{1, \cdots, 5\}$. This information can also be obtained from the command

```
> classlabels(y)

[1] 1 2 3 4 5
attr(,''type'')
[1] ''num''
```

The attribute type "num" indicates that here the class labels realize numeric rather than character ("chr") values.

The length of each bar in Fig. 15 represents the fraction of test set observations in the corresponding class that were misclassified. We see that class 3 was the most difficult to separate, whereas classes 1 and 5 had relatively low error rates.

The next step is to examine the specific nature of the misclassifications. The command

```
> classerrors(class=3)
```

produces the barplot shown in Fig. 16. Each bar represents one of the classes into which class 3 observations were misclassified. The length of the bar is the fraction of class 3 (test set) observations assigned to that (incorrect) class. Here we see that most class 3 observations were misclassified as class 2 and a slightly smaller number as class 4.

One can examine the misclassification structure for each of the other classes in an analogous manner. Figure 17 collects the results for all the classes of this example.

As with regression one can plot the estimated importance of the respective predictor variables with the command

```
> varimp()
```

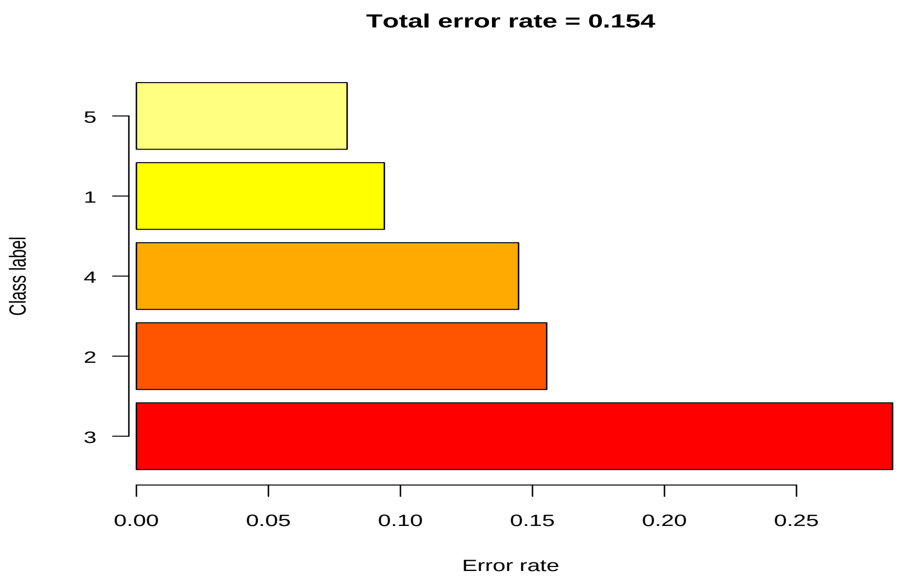

Figure 15: Output of the command `classerrors()`.

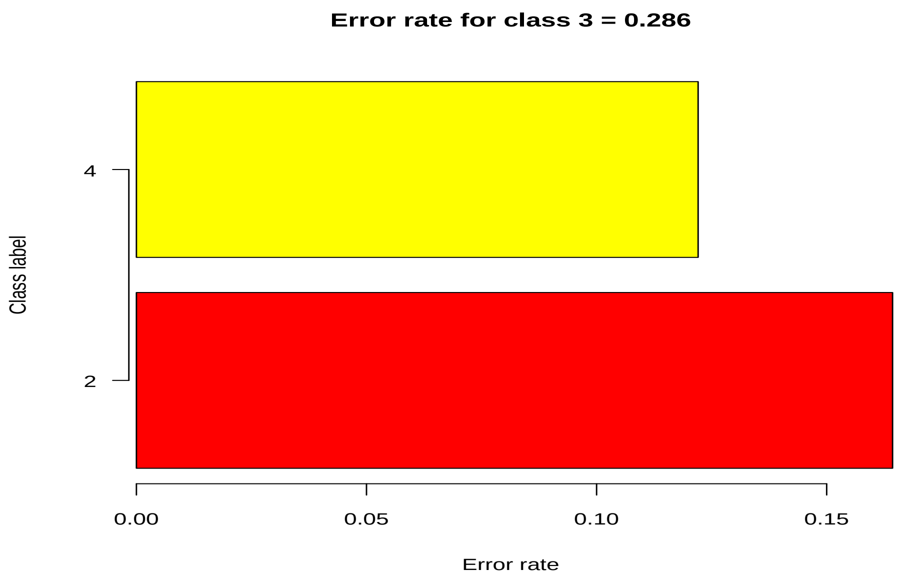

Figure 16: Output of the command `classerrors(class=3)`.

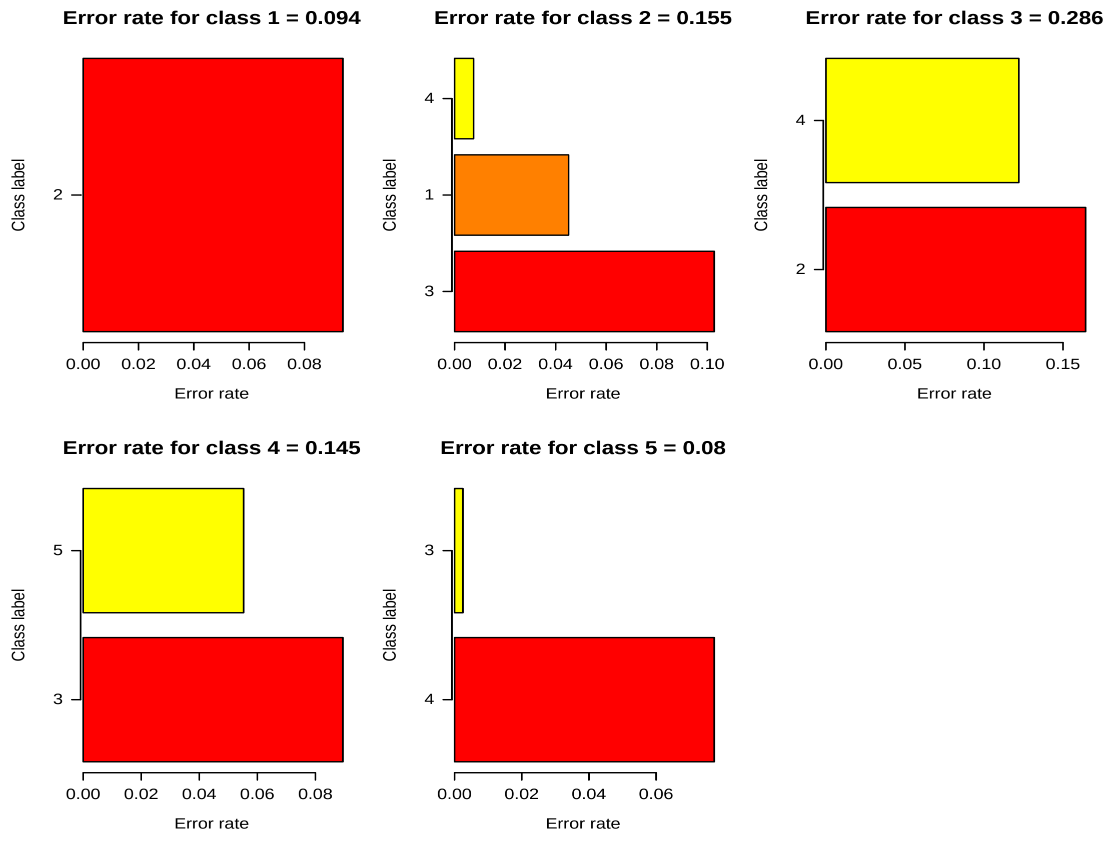

Figure 17: Output of the commands `classerrors(class=1)`; $\cdots$; `classerrors(class=5)`.

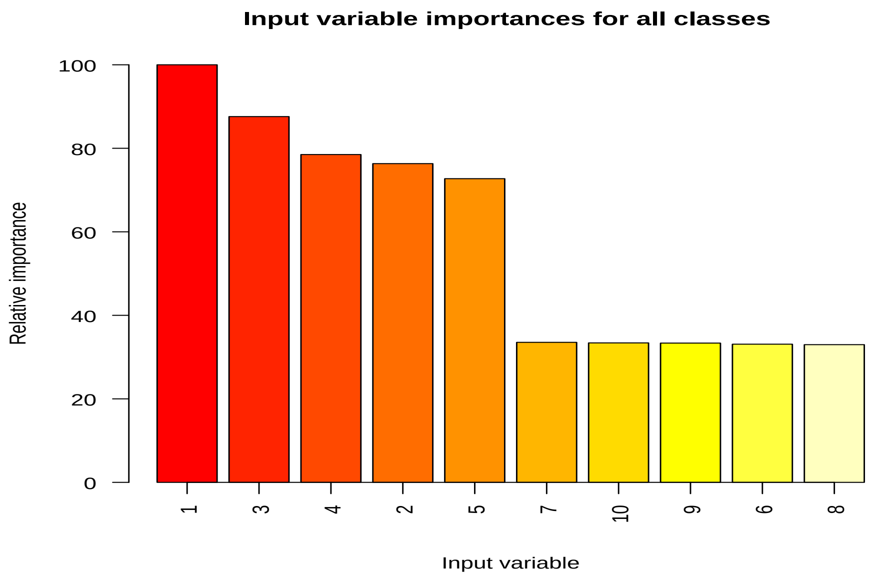

Figure 18: Output of the command `varimp()` for classification.

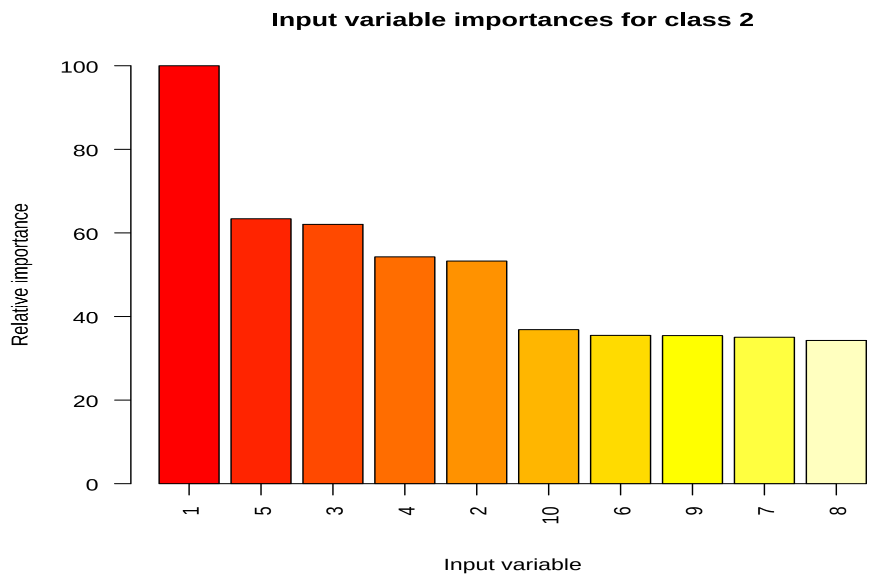

Figure 19: Output of the command `varimp(class=2)`.

producing the plot shown in Fig. 18. This shows the average importance of each of the variables in predicting all of the classes.

One can also see the relative importances of the predictor variables in separating each of the *individual* classes. The command

```
> varimp(class=2)
```

produces the plot shown in Fig. 19. Comparing this plot with Fig. 18, one sees that x_1 is relatively more important in separating class 2 than it is averaged over all of the classes.

Given a class label, *varimp* allows one to identify which predictor variables are most important in separating the corresponding class from the other classes. Sometimes it may be of interest to pose the complement question. Given a predictor variable, to which classes does it contribute most to their separation. This can be done with the command

```
> classimp(var=2)
```

producing the plot shown in Fig. 20. Here one sees that x_2 contributes considerably more towards separating class 5 than in separating the other classes.

After identifying those predictor variables that are most relevant to separating each of the respective classes, it can be informative to see the nature of the dependence of each class on those variables. The MART model for classification consists of a set of real valued functions $\{f_k(\mathbf{x})\}_{k \in K}$, one for each class label k in the complete set of labels K. Each function is related to the probability $p_k(\mathbf{x}) = \Pr[y = k \,|\, \mathbf{x}]$ by

$$f_k(\mathbf{x}) = \log p_k(\mathbf{x}) - \frac{1}{\#K} \sum_{l \in K} \log p_l(\mathbf{x}) \tag{8}$$

where $\#K$ is the total number of classes. Thus, larger values of $f_k(\mathbf{x})$ correspond to increased probability of realizing the class label k for the set of predictor values $\mathbf{x}$. Partial dependence plots of $f_k(\mathbf{x})$ on the predictor variables most influential in separating that class can sometimes

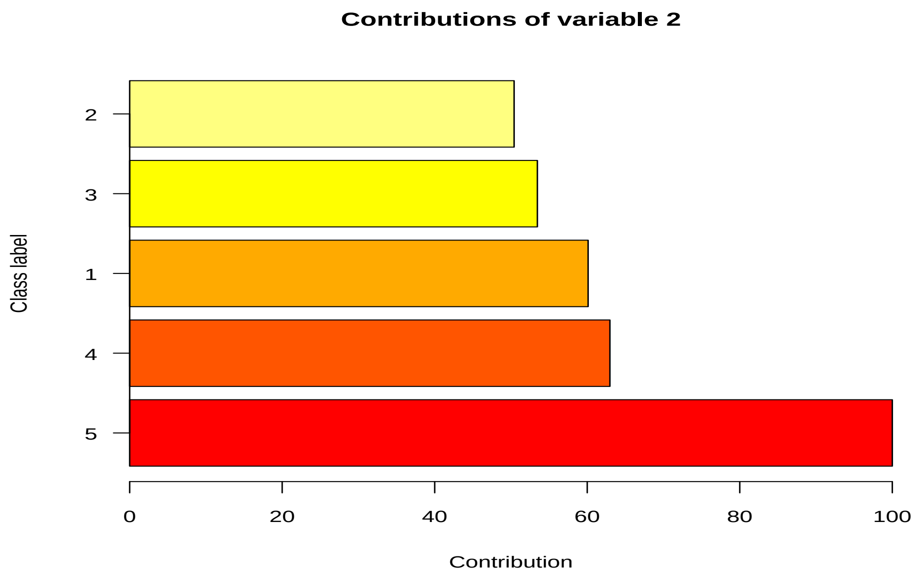

Figure 20: Output of the command `classimp(var=2)`.

provide insight into what values of those variables tend to increase or decrease the odds of observing that class.

As in regression, partial dependence plots are made with *singleplot* and *pairplot*. The command

```
> singleplot(1,class=5)
```

plots the partial dependence of $f_5(\mathbf{x})$ (8) on x_1, as shown in Fig. 21.

The command

```
> pairplot(1,3,class=5)
```

plots the partial dependence of $f_5(\mathbf{x})$ (8) on joint values of x_1 and x_3, as shown in Fig. 22.

The vague resemblance of Figs. 21 and 22 to Figs. 5 and 8 respectively is not a coincidence. The class labels in the file "classlab", used here for illustration, were obtained by partitioning the values of the "target" function (5) at their 20%, 40%, 60%, and 80% points. Class labels were assigned the to the data within each successive interval in ascending values from 1 to 5.

4.2 Prediction

The classification MART model can be used to predict either a class label, or class probabilities, given one or more sets of predictor variable values $\{x_1, x_2, \cdots, x_n\}$. Class label prediction is done using *martpred* as described in Section 3.2. The only difference is that for classification the procedure returns a predicted class label of the same type as "y" (numeric or character) that was input to *mart*. For example

```
> xp_c(0.5,0.5,1,2,rep(0.5,6))

> martpred(xp)
```

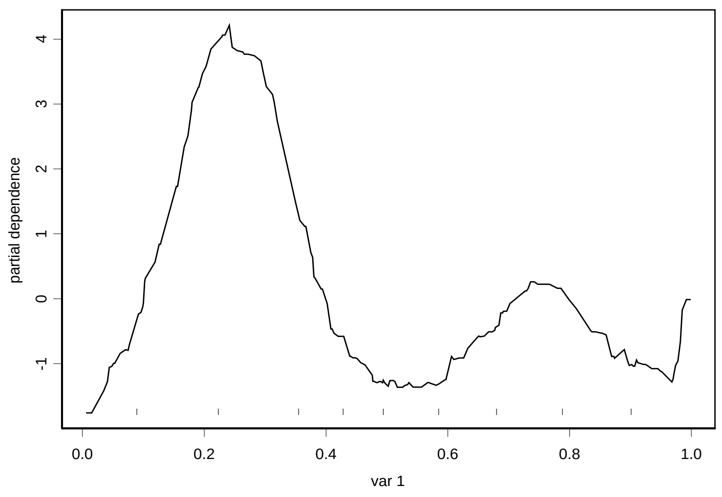

Figure 21: Output of the command `singleplot(1,class=5)`.

```
[1] 3
```

predicts class 3 for the indicated vector “xp”.

Probability prediction is also done with *martpred*

```
> martpred(xp, probs=T)
```

```
[1] 0.001129717 0.203446046 0.448028862 0.330960423 0.016434953
```

The estimated probabilities are reported in the order of their internal *mart* numeric codes. These can be obtained with the command

```
> classlabels(y)
```

```
[1] 1 2 3 4 5
attr(,''type'')
[1] ''num''
```

which returns the class labels in internal *mart* order.

If “xp” is an *nobs* by *nvar* matrix representing *nobs* observations to be predicted, *martpred* returns a vector of predicted labels of length *nobs*, or an *nobs* by *nclass* matrix of probabilities (probs=T). Here *nvar* is the number of predictor variables (columns of “x”), and *nclass* is the number of classes (distinct values of y).

5 Saving the MART model

Each *mart* or *moremart* command causes the resulting model to be written to an internal working storage buffer, over writing the model currently stored there. All subsequent MART commands associated with this model (*moremart, progress, martstat, martpred, varimp, singleplot, pairplot, classimp, classerrors*) reference this buffer. The *martsave* command can be used to save the contents of the current working buffer. It can then be retrieved in its present state at a later time for further analysis or to make predictions. For example,

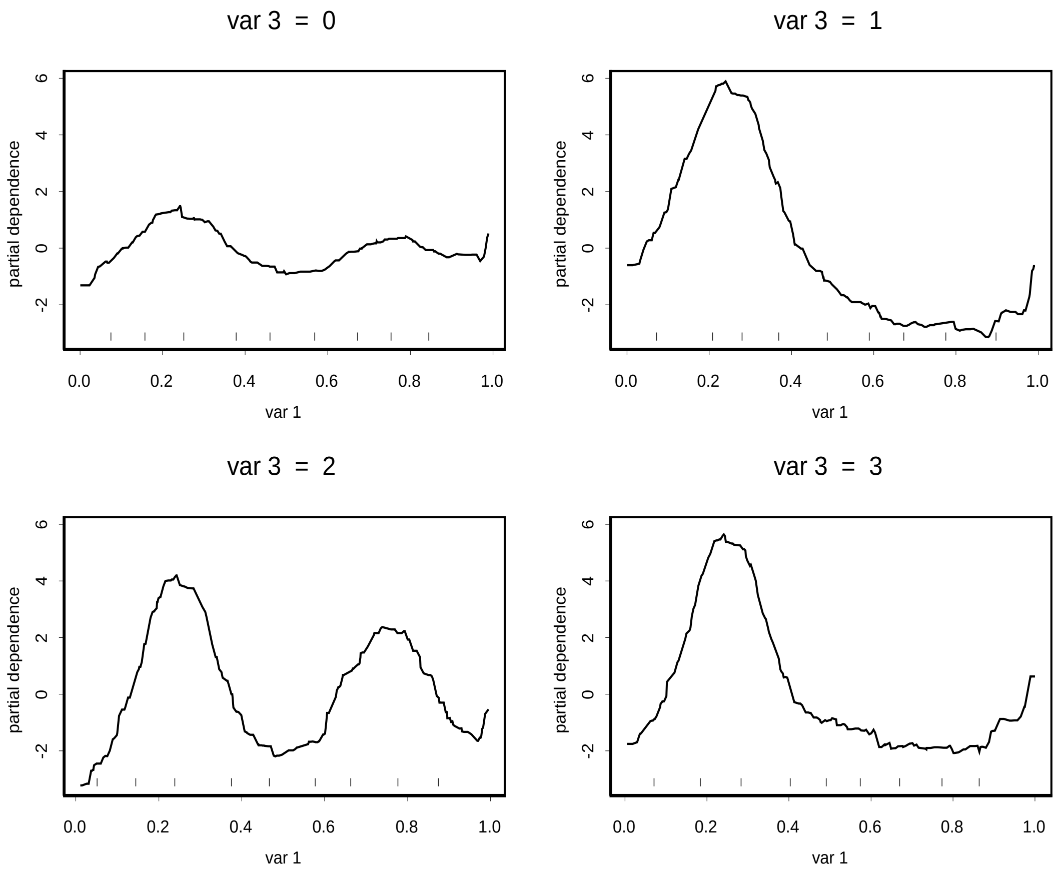

Figure 22: Output of the command `pairplot(1,3,class=5)`.

```
 > martsave('tutorial')

 OK
```

saves the MART model developed for the present illustrative example under the name "tutorial". If that name is the same as a currently saved model, the latter will be overwritten by the current buffer.

The command *martget* transfers a previously saved model to the current working buffer. It can then be analyzed in the same manner as before it was saved (with *martsave*). For example,

```
 > martget('tutorial')

 OK
```

can be used to retrieve the illustrative example at a later time. Note that this over writes the model stored in the current working buffer, which should be saved (if needed) before issuing the *martget* command.

The command *martlist* prints a list of all currently saved MART models:

```
 > martlist()

 $tutorial
```

The leading character (here $) is for internal use and should be ignored when referencing the corresponding saved models.

Finally, the *marterase* command can be used to irrevocably remove a previously saved MART model:

```
 > marterase('tutorial')

OK

 > martlist()

No MART models saved
```

References

[1] Friedman, J. H. (1999a). Greedy function approximation: a gradient boosting machine. http://www-stat.stanford.edu/~jhf/ftp/trebst.ps

[2] Friedman, J. H. (1999b). Stochastic gradient boosting.
http://www-stat.stanford.edu/~jhf/ftp/stobst.ps