

Stat 8053, Fall 2011: Recursive Partitioning

The objective of these data is to infer a sensible pricing model for diamonds based on data pertaining to their weight (in carats), their color (either D, E, F, G, H or I), clarity (either IF, VVS1, VVS2, VS1 or VS2) and the certification agency (either GIA, IGI or HRD).

Source: Chu, Singfat (2001), Pricing the C's of Diamond Stones, *Journal of Statistics Education Volume 9*, Number 2 (2001), <http://www.amstat.org/publications/jse/v9n2/datasets.chu.html>.

```
> loc <- "http://www.stat.umn.edu/~sandy/courses/8053/Data/diamonds.txt"
> diamonds <- read.table(url(loc), header = TRUE)
> dim(diamonds)

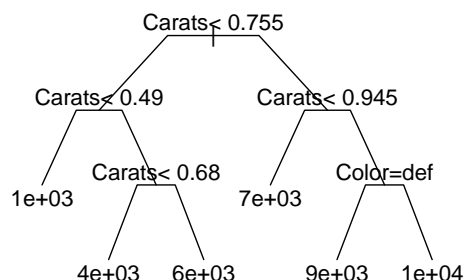
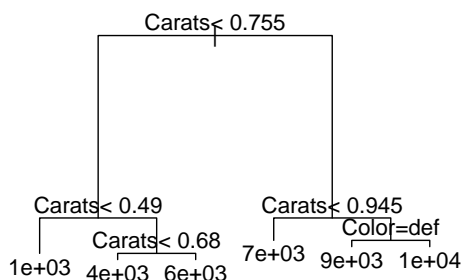
[1] 308  5
```

I will do the fitting with 200 cases and use 108 for validation.

```
> set.seed(10131985)
> est.set <- sample(1:308, 200)
> val.set <- (1:308)[-est.set]
> diamonds1 <- diamonds[est.set, ]
```

Fit using `rpart`, which implements the methodology in L Breiman, J. Friedman, R. Olshen and C. Stone (1984) *Classification and Regression Trees*, Wadsworth.

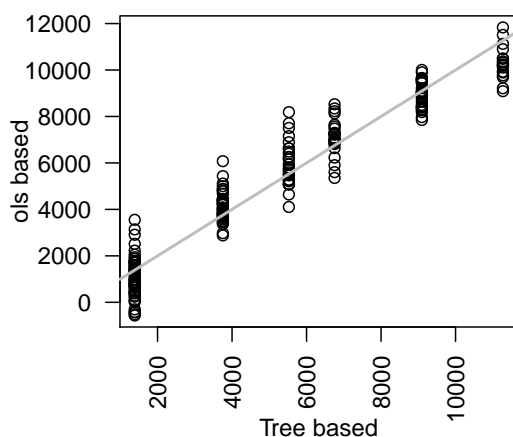
```
> library(rpart)
> r1 <- rpart(Price ~ ., diamonds1, method = "anova")
> par(mfrow = c(1, 2), xpd = NA)
> plot(r1, margin = 0.1)
> text(r1)
> plot(r1, compress = TRUE, uniform = TRUE, branch = 0.4, margin = 0.1)
> text(r1)
```



The `method = "anova"` specifies the criterion for the splitting, with `anova` presumably suggesting minimizing within terminal node sum of squared errors. The variable `Color` appears to be relevant only if `Carats > 0.945` and thus the tree model “automatically” included an interaction. One version of tree-building is even called *automatic interaction detection*.

Compare the fit of `rpart` to `ols` fitting:

```
> m1 <- lm(Price ~ ., diamonds1)
> plot(predict(m1) ~ predict(r1), xlab = "Tree based", ylab = "ols based",
+       las = 2)
> abline(0, 1, col = "gray", lwd = 2)
```



Tree pruning

Expand `rpart` output to trees that are certainly too big and then look at the fit statistics. If RSS_i is the sum of squared errors at terminal node i , then the *cost-complexity* criterion is

$$CC(\lambda) = \sum RSS_k + \lambda \times \text{number of terminal nodes}$$

where λ is like a smoothing parameter, with large λ favoring small trees and small λ favoring big trees. $CC(\lambda)$ is estimated using 10-fold generalized cross validation, meaning that 10% of the data are deleted; the model is fit to the remaining 90%, and error is estimated for the 10% deleted. This is repeated for each 10% of the data, and then averaged. Both the textbook and `rpart` use the statistic

$$cp = \lambda / \sim \sum (y_{\ell i} - \bar{y}_{\ell})^2$$

as the smoothing parameter, where the numerator is the sum of squares errors when the tree has one node and no branches, and represents the worst case possible. This can be estimated from the 10-fold cross validation.

```
> r2 <- update(r1, cp = 0.001, xval = 10)
> printcp(r2)
```

Regression tree:

```
rpart(formula = Price ~ ., data = diamonds1, method = "anova",
      cp = 0.001, xval = 10)
```

Variables actually used in tree construction:

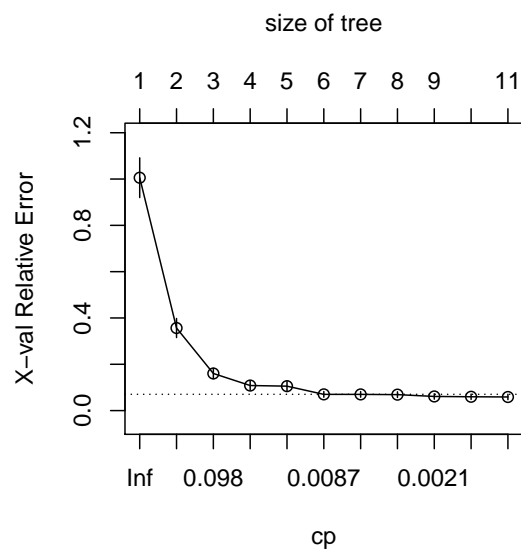
```
[1] Carats Color
```

Root node error: $2.2e+09/200 = 1.1e+07$

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.6931	0	1.000	1.012	0.086
2	0.1346	1	0.307	0.370	0.039
3	0.0717	2	0.172	0.202	0.027
4	0.0239	3	0.101	0.113	0.019
5	0.0237	4	0.077	0.109	0.019
6	0.0032	5	0.053	0.074	0.012
7	0.0030	6	0.050	0.071	0.012
8	0.0027	7	0.047	0.071	0.012
9	0.0016	8	0.044	0.066	0.012
10	0.0015	9	0.042	0.066	0.012
11	0.0010	10	0.041	0.062	0.012

```
> plotcp(r2)
```

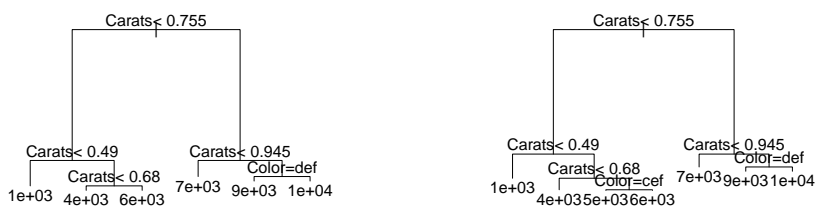


The argument `cp=0.001` is passed from `update` to `rpart` to `rpart.control`, where it is used to determine the stopping rule for the tree construction. The default is to stop if the next partition has a value of `cp` less than 0.01. We set it to a smaller value to allow larger trees to be built. The argument `xval` is the number of splits used in generalized cross validation, with default value of 10.

The functions `printcp` and `plotcp` display summaries at each partition. The quantities in the `cp` output are the value of `cp`; `nsplit` = number of terminal nodes; `rel error` = $\sum \sum (y_{li} - \bar{y}_l)^2 / \sum \sum (y_{li} - \bar{y})^2$; `xerror` = like the relative error but based on k -fold GCV. `xstd` is the SD for the k splits.

Here are the pruned tree and the original tree:

```
> par(mfrow = c(1, 2), xpd = NA)
> r3 <- prune(r2, cp = r2$cptable[7, 1])
> plot(r1, margin = 0.1)
> text(r1)
> plot(r3, margin = 0.1)
> text(r3)
```



How well does this work? Let's compute in-sample and validation sample errors for the tree, for ols, and for stepwise regression:

```
> get.error <- function(m, which) {
+   with(diamonds, sqrt(sum((Price[which] - predict(m, diamonds[which,
+     ]))^2)/length(which)))
+ }
> summary(m2 <- step(m1, trace = 0))$coef
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	28.58	283.6	0.1008	9.198e-01
Carats	12435.07	199.7	62.2662	6.791e-128
ClarityVS1	-1334.36	179.3	-7.4430	3.376e-12
ClarityVS2	-1674.92	193.5	-8.6543	2.144e-15
ClarityVVS1	-570.52	187.3	-3.0463	2.648e-03
ClarityVVS2	-1110.86	179.6	-6.1867	3.722e-09
ColorE	-1006.68	261.6	-3.8475	1.631e-04
ColorF	-1675.97	251.2	-6.6710	2.729e-10
ColorG	-1894.43	253.3	-7.4799	2.719e-12
ColorH	-2480.34	254.5	-9.7446	1.948e-18
ColorI	-2932.57	262.2	-11.1849	1.295e-22

```
> (ans <- data.frame(rpart = c(get.error(r3, est.set), c(get.error(r3,
+   val.set))), ols = c(get.error(m1, est.set), c(get.error(m1,
```

```
+ val.set))), ols.step = c(get.error(m2, est.set), c(get.error(m2,
+ val.set))), row.names = c("Estimation", "Validation"))
```

```
      rpart  ols ols.step
Estimation 745.4 662.7   665.5
Validation 881.0 786.1   782.9
```

Classification

As an example we will use the blowdown data from an earlier handout. The response is equal to one if the tree died and zero if it survived. If the response is a factor, then the algorithm does classification, not regression.

Squared error loss is not appropriate for classification, and by default `rpart` uses the *Gini index* as a measure of lack of fit. If we let p_{ik} be the fraction of points in node i that fall in response category k , then $D_i = 1 - \sum p_{ik}^2$ is a measure of lack-of-fit. $D_i = 0$ if one of the $p_{ik} = 1$ and all the rest are zero, and for a binary response $D_i = 1/2$ when $p_{i1} = p_{i2} = 1/2$. Other choices of purity measures are apparently available in `rpart` but I haven't been able to figure out how to get them with the terrible documentation that is available.

```
> library(alr3)
> par(mfrow = c(1, 2), xpd = NA)
> set.seed(123456)
> const <- rep(FALSE, dim(blowdown)[1])
> const[sample(1:dim(blowdown)[1], 2000)] <- TRUE
> blowdown$y1 <- factor(blowdown$y)
> r3 <- rpart(y1 ~ S + D + SPP, blowdown, subset = const, method = "class")
> plot(r3, compress = F, uniform = T, branch = 0.5, margin = 0.2)
> text(r3)
> r4 <- update(r3, cp = 0.001)
> plotcp(r4)
> printcp(r4)
```

Classification tree:

```
rpart(formula = y1 ~ S + D + SPP, data = blowdown, subset = const,
      method = "class", cp = 0.001)
```

Variables actually used in tree construction:

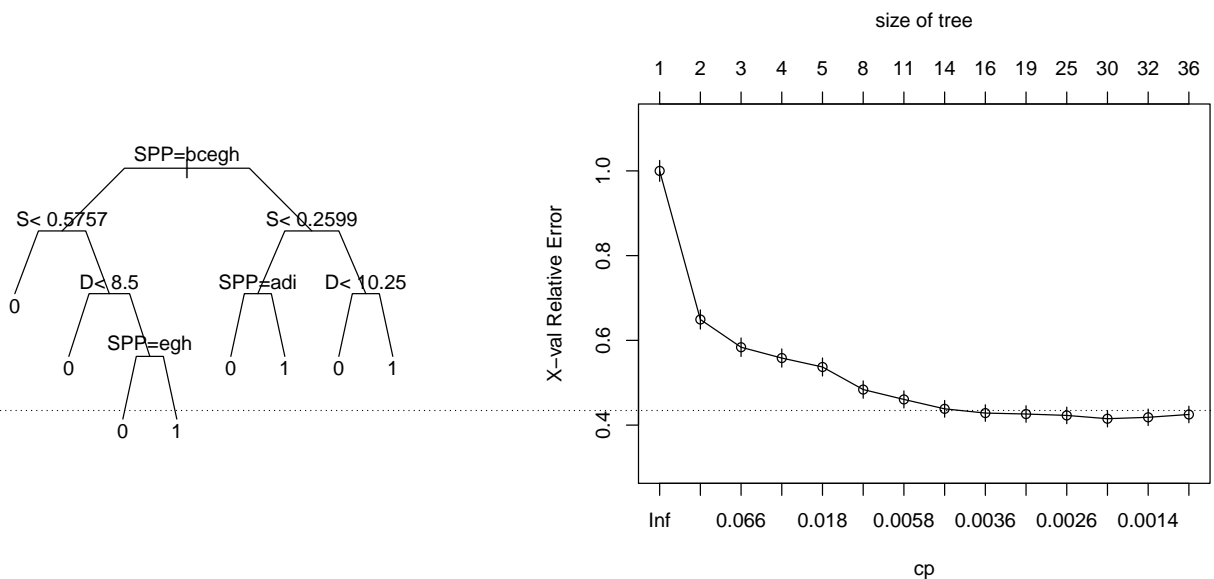
```
[1] D   S   SPP
```

Root node error: 901/2000 = 0.45

n= 2000

```
      CP nsplit rel error xerror  xstd
1 0.3507      0      1.00   1.00 0.025
```

2	0.1099	1	0.65	0.65	0.023
3	0.0400	2	0.54	0.58	0.022
4	0.0200	3	0.50	0.56	0.022
5	0.0166	4	0.48	0.54	0.021
6	0.0081	7	0.42	0.48	0.020
7	0.0041	10	0.40	0.46	0.020
8	0.0039	13	0.39	0.44	0.020
9	0.0033	15	0.38	0.43	0.020
10	0.0030	18	0.37	0.43	0.020
11	0.0022	24	0.35	0.42	0.019
12	0.0017	29	0.34	0.42	0.019
13	0.0011	31	0.33	0.42	0.019
14	0.0010	35	0.33	0.43	0.020



A summary of fit for a classification problem is to count correct and incorrect classifications. We do this for the construction and for the validation sets.

The `predict` method for `rpart` will produce a prediction for each category in the response. Thus, with 2000 observations and a binary response, the `predict` method will return the probability associated with each category in a 2000×2 matrix. If you use `predict(r3, type="class")`, it will produce a vector of length 2000 with the name of the category of highest probability for each point.

```
> show <- function(tt) {
+   print(tt)
+   cat(paste("Misclassification rate =", round(1 - sum(diag(tt))/sum(tt),
+     2), "\n"))
+   invisible()
+ }
> show(with(blowdown, table(actual = y1[const], predicted = predict(r3,
+   type = "class"))))
```

```

      predicted
actual  0   1
      0 954 145
      1 237 664
Misclassification rate = 0.19

```

```

> show(with(blowdown, table(actual = y1[!const], predicted = predict(r3,
+   newdata = blowdown[!const, ], type = "class"))))

```

```

      predicted
actual  0   1
      0 764 119
      1 226 557
Misclassification rate = 0.21

```

For comparison, let's do the same thing with the additive model and with the glm model discussed in the last homework.

```

> library(mgcv)
> m3 <- gam(y ~ s(S) + log2(D) + SPP, data = blowdown, family = binomial,
+   subset = const)
> show(with(blowdown, table(actual = y[const], predicted = ifelse(predict(m3) <=
+   0, 0, 1))))

```

```

      predicted
actual  0   1
      0 916 183
      1 204 697
Misclassification rate = 0.19

```

```

> show(with(blowdown, table(actual = y[!const], predicted = ifelse(predict(m3,
+   newdata = blowdown[!const, ]) <= 0, 0, 1))))

```

```

      predicted
actual  0   1
      0 740 143
      1 179 604
Misclassification rate = 0.19

```

```

> m4 <- glm(y ~ S + log2(D) + SPP, data = blowdown, family = binomial,
+   subset = const)
> show(with(blowdown, table(actual = y[const], predicted = ifelse(predict(m4) <=
+   0, 0, 1))))

```

```

      predicted
actual  0   1
      0 914 185
      1 211 690
Misclassification rate = 0.2

```

```
> show(with(blowdown, table(actual = y[!const], predicted = ifelse(predict(m4,
+   newdata = blowdown[!const, ]) <= 0, 0, 1))))

      predicted
actual  0    1
      0 736 147
      1 181 602
Misclassification rate = 0.2
```

Bagging

Recursive partitioning solutions are rather fragile because only one split is considered at a time. Thus, small perturbations in the data are likely to lead to completely different solutions. One possible way to deal with this problem is to recompute the tree based on slightly perturbed data, and then average over the perturbations. Leo Breiman proposed this idea, and called it bootstrap aggregation, or *bagging*.

Here is the general idea. We start with data $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ from which we can estimate a function $\hat{f}(x)$ that can return predictions at the point x . For example, $\hat{f}(x)$ can be from regression tree, an additive model or almost any other fitting procedure. Suppose that \mathcal{D}^b is a bootstrap sample of size n from \mathcal{D} , and $\hat{f}^b(x)$ is the corresponding prediction. Breiman's idea was to replace $\hat{f}(x)$ by $E(\hat{f}^b(x))$, the average over all possible bootstrap samples, as a more stable prediction. In practice this is estimated by

$$\hat{f}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

the average of B bootstrap samples.

The *cost* of bagging is that there is no longer a simple tree representation of the result, as predictions are now done by averaging B trees which can have different complexities, number of nodes, and so on.

Bagging is implemented in the package `ipred`.

```
> library(ipred)
> b1 <- bagging(y1 ~ S + D + SPP, blowdown, subset = const, nbagg = 100)
> show(with(blowdown, table(actual = y[const], predicted = predict(b1))))

      predicted
actual  0    1
      0 919 180
      1 215 686
Misclassification rate = 0.2

> show(with(blowdown, table(actual = y[!const], predicted = predict(b1,
+   newdata = blowdown[!const, ]))))

      predicted
actual  0    1
      0 724 159
      1 200 583
Misclassification rate = 0.22
```

In this example bagging had little effect, probably because sample size is so large.

Random Forests

Random forests are an elaboration of bagging due to Breiman in which each tree is grown using an algorithm that is more complicated than simply using recursive partitioning. The package `randomForest` is used for the fitting and includes some interesting summary graphs. According to Breiman's website, here is a description of Random Forests:

1. If the number of cases in the training set is N , sample N cases at random with replacement, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m \ll M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.
4. Average over all the trees.

```
> library(randomForest)
> b2 <- randomForest(y1 ~ S + D + SPP, blowdown, subset = const)
> show(with(blowdown, table(actual = y1[const], predicted = predict(b2))))
```

```
      predicted
actual  0    1
      0 926 173
      1 203 698
```

Misclassification rate = 0.19

```
> show(with(blowdown, table(actual = y1[!const], predicted = predict(b2,
+   newdata = blowdown[!const, ]))))
```

```
      predicted
actual  0    1
      0 749 134
      1 178 605
```

Misclassification rate = 0.19

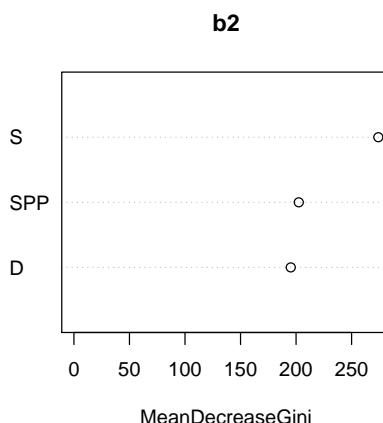
Variable importance

Consider predictor ℓ and a particular node. Let $I(v(t) = \ell)$ be an indicator function whose value is 1 if the t -th node in the tree is a split on variable ℓ , and equal to zero otherwise. Let \hat{i}_t^2 be the decrease in risk (generally squared error loss, but any loss function you like) for including node $v(t)$. Then the importance of variable ℓ in the tree is

$$I_\ell^2 = \sum_{i=1}^{J-1} \hat{i}_t^2 I(v(t) = \ell)$$

where J is the number of nodes in the tree. For many trees, average I over all the trees.

```
> varImpPlot(b2, type = 2)
```



Partial Dependence Plots

Suppose that X_k is a variable of interest, and $X_{\setminus k}$ is the vector of remaining predictors. Consider the problem of display of the dependence of the response on X_k that is implied by the model. If the true mean function is additive

$$E_T(Y|X_k, X_{\setminus k}) = h_1(X_k) + h_2(X_{\setminus k}) \quad (1)$$

then a plot of $\widehat{h}_1(X_k)$ versus X_k , as is done in the `mgcv` package is the appropriate plot. This of course assumes that we have an estimated the function h_1 , but in methods that average over models like random forests, bagging or other model aggregation methods, we may not have an explicit estimate for h_1 . If the true mean function is not additive,

$$E_T(Y|X_k, X_{\setminus k}) = h_1(X_k) + h_2(X_{\setminus k}) + h_{12}(x_k, x_{\setminus k}) \quad (2)$$

then the idea that there is an “effect” due to X_1 alone is in doubt, and conditioning on the value of $x_{\setminus k}$ may be required to get a summary graph. We are used to these ideas in the *interaction graphs* that are commonly used with factorial models. These graphs are elegantly implemented in the `effects` package in R.

Marginal Model Plots

Recall that a marginal model plot displays Y versus X_k with two smooths. The smoother fit to the points estimates $E_T(Y|X_k)$, the marginal expectation of $Y|X_k$. The second smooth is to \hat{Y} versus X_k , and this smooth will estimate $E_{X_{\setminus k}}[E(Y|X_k, X_{\setminus k})] \approx E_T(Y|X_k)$ when the fitted model is correct, but it may not do so otherwise, and so comparison of the two smooths gives information about the fit of the marginal-model, but it does not directly say anything about the dependence of Y on X_k in $E_T(Y|X_k, X_{\setminus k})$.

Partial Residual Plots

These plots are variously called partial residual plots, component plus residuals plots and are a special case of CERES plots. If you have a linear model $E_T(Y|X_k, X_{\setminus k}) = \beta_0 + \beta_k X_k + \beta'_{\setminus k} X_{\setminus k}$, then this is a plot of X_k versus $\hat{e} + \hat{\beta}_k X_k$, where \hat{e} are the ordinary residuals. This is essentially the same as the plots produced by `mgcv`.

If the true mean function is additive like (1) and we average over the distribution of $X_{\setminus k}$, we get

$$E_{X_{\setminus k}} [E_T(Y|X_k, X_{\setminus k})] = h_1(X_k) + E_{X_{\setminus k}} [h_2(X_{\setminus k})]$$

which differs from the partial residual plot only by a location factor.

In the more general case of (2) we get

$$E_{X_{\setminus k}} [E_T(Y|X_k, X_{\setminus k})] = h_1(X_k) + E_{X_{\setminus k}} [h_2(X_{\setminus k})] + E_{X_{\setminus k}} [h_{12}(X_k, X_{\setminus k})]$$

and so in this case the partial residual plot is not the right graph in general. If we could estimate the last term in this equation, which is possible in parametric models, we could examine plots somehow that condition of the value of $X_{\setminus k}$.

Partial Response Plot

Friedman, (2001 *Annals of Stats*, 29, 1189–1232) suggested a plot that could be used to summarize the dependence of a response on a particular predictor even if the model is a black box like a random forest. The suggested plot is of

$$X_k \text{ versus } \frac{1}{m} \sum_{i=1}^m \hat{E}(Y|X_k, X_{\setminus k} = x_{i,\setminus k})$$

where the $x_{i,\setminus k}$ are appropriately chosen. The usual default is the rows of $X_{\setminus k}$, the observed values of these predictors, but other choices are possible. Under (1) this is equivalent to the partial residual plot if m is large enough and the $X_{\setminus k}$ are sampled from their marginal distribution. Under (2) it suffers the same interpretability problem as the partial residual plot.

If the true mean function is of the form

$$E_T(Y|X_k, X_{\setminus k}) = h_1(X_k) + h_2(X_{\setminus k})$$

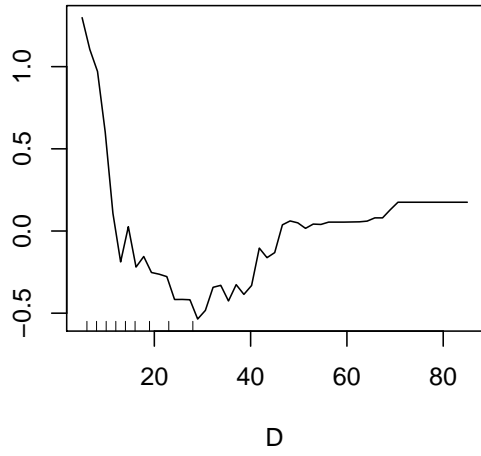
averaging is irrelevant and apart from a location factor this plot will display $h_1(X_k)$. If

$$E_T(Y|X_k, X_{\setminus k}) = h_1(X_k) + h_2(X_{\setminus k}) + h_{12}(x_k, x_{\setminus k})$$

the average is different and can be more informative. This does not correspond to the marginal model plot.

```
> par(mfrow = c(1, 2), xpd = NA)
> partialPlot(b2, blowdown, D)
> partialPlot(b2, blowdown, S)
```

Partial Dependence on D



Partial Dependence on S

