Tree Based Methods

Lu Haibo

Monday, Jun 8, 2015

Introduction

Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the other supervised learning approaches in terms of prediction accuracy.

However, by producing multiple trees which are then combined to yield a single consensus prediction. We will see that combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation.

The Basics of Decision Trees

```
set.seed(1)
library(ISLR)
data(Hitters)
library(rpart.plot)

fit <- rpart(log(Salary) ~ Years + Hits, data = Hitters)
fit.prune <- prune(fit, cp = 0.1)
rpart.plot(fit.prune)
```

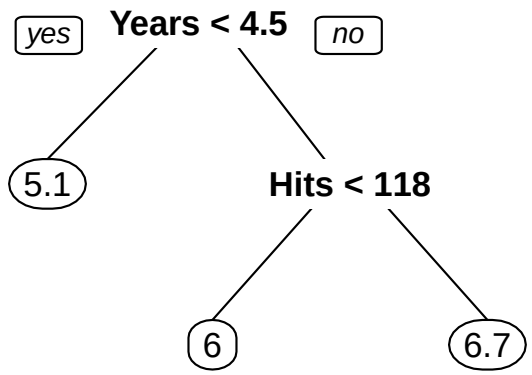
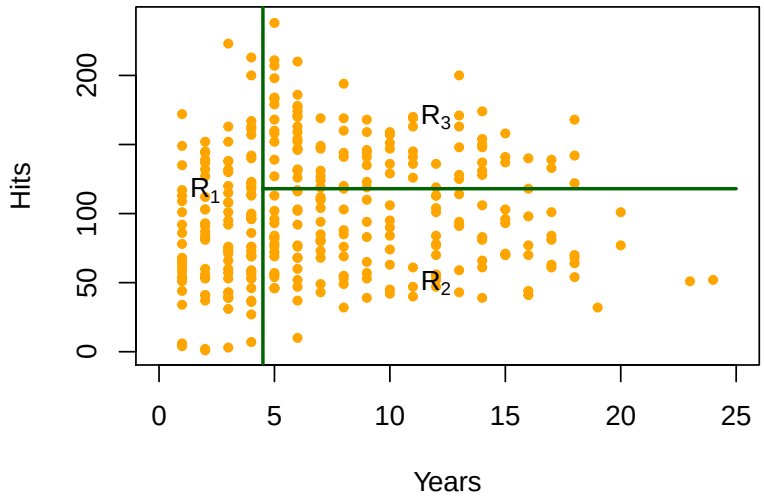


Figure 1: For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to $Years < 4.5$, and the right-hand branch corresponds to $Years \geq 4.5$. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Overall, the tree stratifies or segments the players into three regions of predictor space:

Figure 2: The three-region partition for the Hitters data set from the regression tree illustrated in Figure 1.



- $R_1 = \{X | Years < 4.5\}$
- $R_2 = \{X | Years >= 4.5, Hits < 118\}$
- $R_3 = \{X | Years >= 4.5, Hits >= 118\}$

The predicted salaries for these three groups are $\$1,000 \times e^{5.1} = \$164,022$, $\$1,000 \times e^{6} = \$403,429$, and $\$1,000 \times e^{6.7} = \$812,406$

We might interpret the regression tree displayed in Figure 1 as follows:

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of hits that he made in the previous year seems to play little role in his salary.
- But among players who have been in the major leagues for five or more years, the number of hits made in the previous year does affect salary, and players who made more hits last year tend to have higher salaries.

The regression tree shown in Figure 1 is likely an over-simplification of the true relationship between `Hits`, `Years`, *and* `Salary`. *However, it has advantages over other types of regression models: it is easier to interpret, and has a nice graphical representation.*

Prediction via Stratification of the Feature Space

The process of building a regression tree:

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$ —into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

   The key provlem: **How do we construct the regions** $R_1, \ldots, R_J$?

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangle, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

  where $\hat{y}_{R_j}$ is the mean for the training observations within the jth box.
- Unfortunately, it is computationally infeasible to be consider every possible partition of the feature space into J boxes. For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*. [1]

Recursive Binary Splitting

1. Select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. That is, for any j and s, we define the pair of half-planes

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\},$$

   and we seek the values of j and s that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

2. We repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
3. The process continues until a *stopping criterion* is reached; for instance, we may continue until no region contains more than five observations.

[1] It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

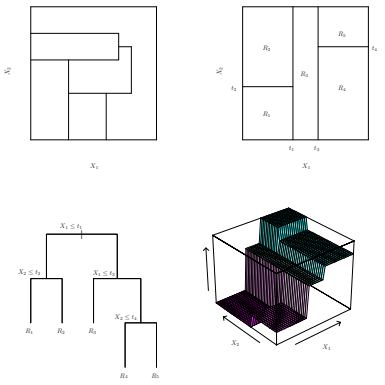


Figure 3: Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting. Top Right: The output of recursive binary splitting on a two-dimensional example. Bottom Left: A tree corresponding to the partition in the top right panel. Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Once the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Tree Pruning

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because *the resulting tree might be too complex*.

A smaller tree with fewer splits (i.e., fewer regions $R_1, \ldots, R_J$) might lead to

- lower variance
- better interpretation
- a little bias

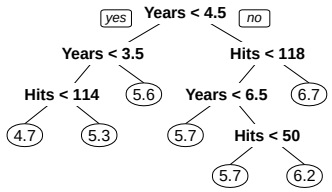


Figure 4: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

How to select a subtree that leads to the lowest test MSE? — Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter λ. For each value of λ, there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{x+i \in R_m} (y_i - \hat{y}_{R_m})^2 + \lambda |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, and T_0 is the origin large tree. [2]

[2] When $\lambda = 0$, then the subtree T will simply equal T_0. However, as λ increases, there is a price to pay for having a treewith many terminal nodes, and so will tend to be a smaller subtree.

```
plotcp(fit)
```

Classification Trees

A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.

For a classification tree, we predict that each observation belongs to *the most commonly occurring class* of training observations in the region to which it belongs.

In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the *class purity*[3] among the training observations that fall into that region.

[3] For example, in a two-class problem with 400 observations in each class(denote this by (400,400)), suppose one split created nodes (300, 100) and (100, 300), while the other created nodes (200, 400) and (200, 0). Both splits produce a misclassification rate of 0.25, but the second split produces a pure node and is probably preferable.

```
set.seed(1)
data(Carseats)
High <- ifelse(Carseats$Sales <= 8, "No", "Yes")
Carseats$High <- as.factor(High)
```

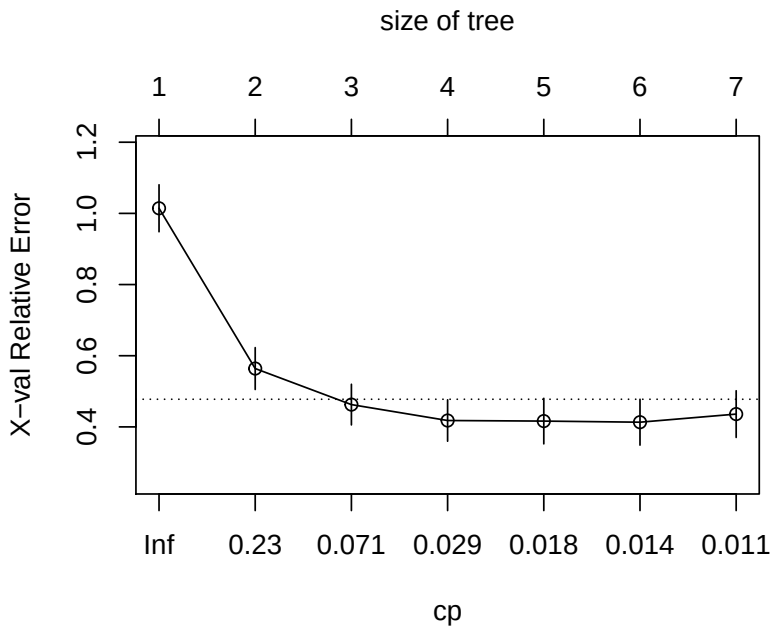


Figure 5: Regression tree analysis for the Hitters data. The cross-validation is shown as a function of the complexity parameter λ or the number of terminal nodes in the pruned tree. The horizontal dashed line is drawn 1SE above the minimum of the curve. A good choice of λ for pruning is often the leftmost value for which the mean lies below the horizontal line.

```
train <- sample(1:nrow(Carseats), nrow(Carseats) *
    0.75)
r.control <- rpart.control(minsplit = 10, cp = 0)
tree <- rpart(High ~ . - Sales, data = Carseats[train,
    ], method = "class", control = r.control)
rpart.plot(tree, extra = 2)


# confusion matrix on the test data
tree.pred <- predict(tree, Carseats[-train, ],
    type = "class")
table(tree.pred, Carseats$High[-train])

##
## tree.pred No Yes
##       No  44   8
##       Yes 15  33

par(mfrow = c(1, 2))
plotcp(tree)
# printcp(fit)
tree.pruned <- prune(tree, cp = 0.02)
rpart.plot(tree.pruned, extra = 2)
```

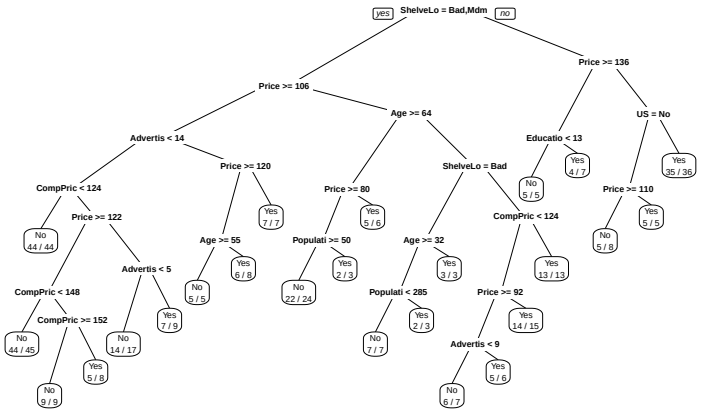


Figure 6: The unpruned tree on the Carseats data set.

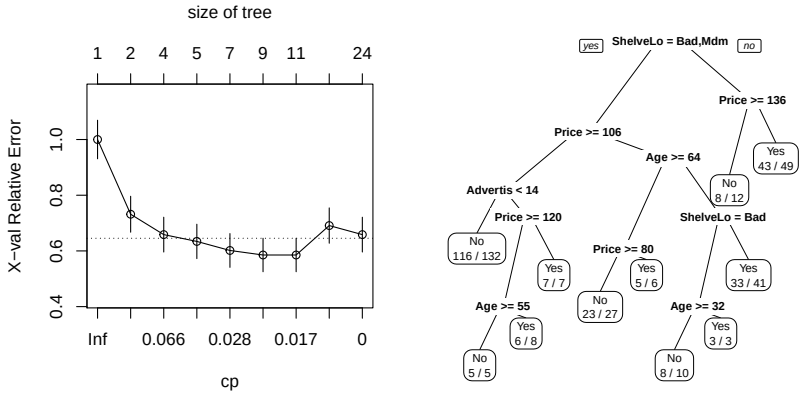


Figure 7: Left: Cross-validation error for different sizes of the pruned tree. Right: The pruned tree with 9 leafs.

```
# confusion matrix on the test data
tree.pruned.pred <- predict(tree.pruned, Carseats[-train,
    ], type = "class")
table(tree.pruned.pred, Carseats$High[-train])
```

```
##
## tree.pruned.pred No Yes
##              No  46  13
##              Yes 13  28
```

Advantages and Disadvantages of Trees

- very easy to explain. In fact, even easier to than linear regression
- more closely mirror hunman decision-making
- can be displayed graphically
- can easily handle qualitative predictors without the need to create dummy variables

Unfortuantely, trees generally do not have the same level of predictive accuracy as other regression and classification approaches.

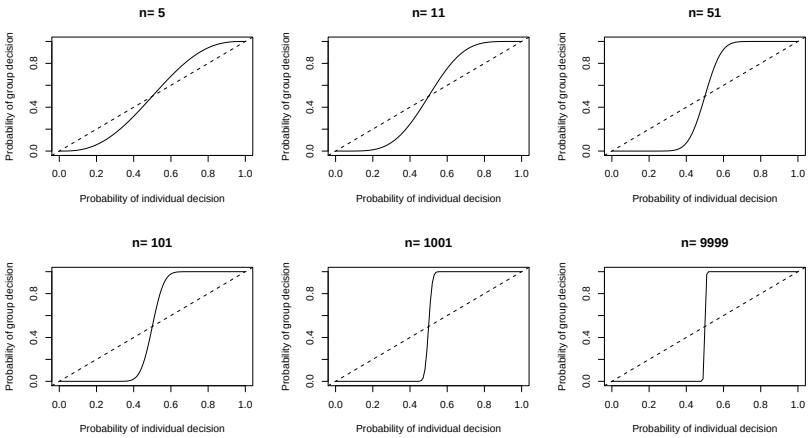
Bagging, Random Forest, Boosting

By aggregating many decision trees, the predictive performance of trees can be substantially improved.

Motivation

Conbines the outputs of many "weak classifiers" to produce a powerful "committee" [4]

[4] Weak classifier: one whose error rate is only slightly better than random guessing



Bagging (Bootstrap Aggregating)

- The decision trees suffer from *high variance*. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- Bagging is a general-purpose procedure for reducing the variance of a statistical learning method.

Bagging:

1. Generate B different boostrapped training data
2. Train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$
3. Average all the predictions, to obtain

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

Out-of-Bag Error Estimation

- There is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach.
- One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations.

In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation.

```
library(randomForest)
bag <- randomForest(High ~ . - Sales, data = Carseats[train,
    ], mtry = 10, ntree = 5000, importance = T)
# confusion matrix by OOB estimate
bag$confusion
```

```
##      No Yes class.error
## No  149  28   0.1581921
## Yes  40  83   0.3252033
```

```
bag.pred <- predict(bag, Carseats[-train, ], type = "class")
# confusion matrix on the test data
table(bag.pred, Carseats$High[-train])
```

```
##
## bag.pred No Yes
##      No  48   9
##      Yes 11  32

varImpPlot(bag, main = "", pch = 20, col = "red",
    lcolor = "darkblue")
```

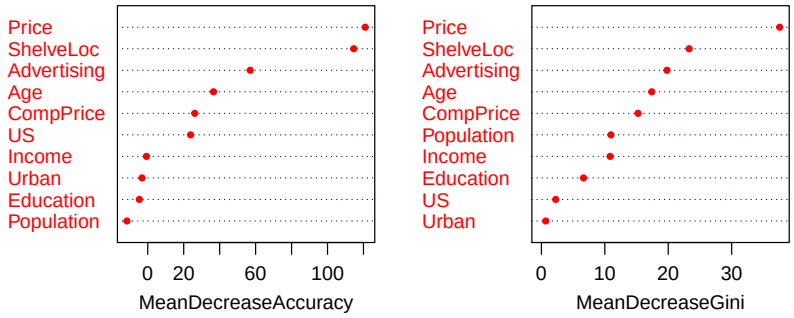


Figure 8: Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using the RSS (for bagging regression tree) or the Gini index (for bagging classification trees).

Random Forests

Random forest provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees.

As in bagging, we build a number of decision trees on bootstrapped training samples. Bu when building these decision trees, each time a split in a tree is considered, a **random sample of m predictors** is chosen as split candidates from the full set of p predictors. [5]

[5] Typically, we choose $m = \sqrt{p}$

Motivation

- In the collection of bagged trees, most or all of the trees will use the strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated.
- Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities. This means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

- Random forests overcome this problem by forcing each split to consider only a subset of the predictors. We can think of this process as *decorrelating the trees*.

```
rf <- randomForest(High ~ . - Sales, data = Carseats[train,
    ], ntree = 5000, importance = T)
# confusion matrix by OOB estimate
rf$confusion
```

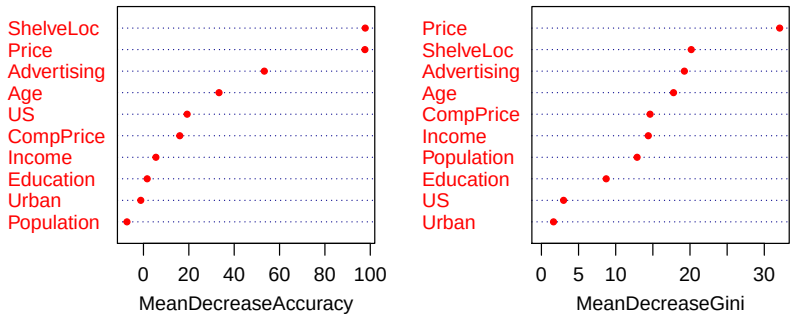
```
##      No Yes class.error
## No  154  23   0.1299435
## Yes  42  81   0.3414634
```

```
rf.pred <- predict(rf, Carseats[-train, ], type = "class")
# confusion matrix on the test data
table(rf.pred, Carseats$High[-train])
```

```
##
## rf.pred No Yes
##     No  49   9
##     Yes 10  32
```

```
varImpPlot(rf, main = "", pch = 20, col = "red",
    lcolor = "darkblue")
```

Figure 9: A variable importance plot for the Carseats data using Random Forests



Boosting

- *Bagging (Random Forest)*:
  - creating multiple copies of the original training data set using the bootstrap

  - fitting a separate decision tree to each copy (each tree is built on a bootstrap data set, independent of the other trees)
  - combining all of the trees in order to create a single predictive model

- *Boosting*:
  - each tree is grown using information from previously grown trees
  - does not involve bootstrap sampling
  - each tree is fit on a modified version of the original data set

---

Algorithm: Boosting for Regression Trees

---

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
(a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
(b) Updata $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

(c) Updata the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

The idea behind boosting: Unlike fitting a single large decision tree to the data, which amounts to *fitting the data heard* and potentially overfitting, the boosting approach instead *learn slowly*

Boosting has three turning parameters:

1. The number of trees B
   - Unlike bagging and random forest, boosting can overfit if B is too large
2. The shrinkage parameter λ
   - Controls the rate at which boosting learns
   - Typical values are 0.01 or 0.001
3. The number d of splits in each tree

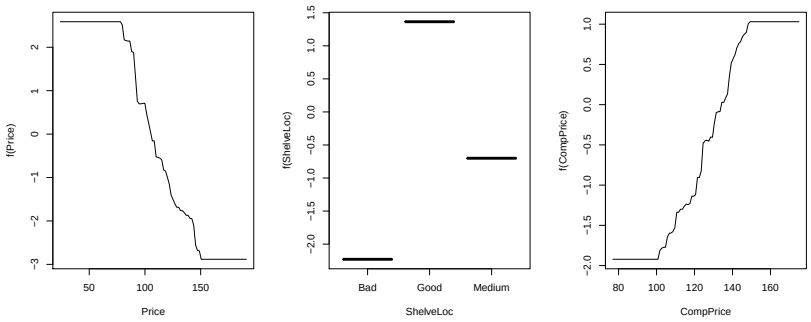
- Controls the complexity of the boosted ensemble.
- Often $d = 1$ works well [6]

[6] in this case, the boosted ensemble is fitting an additive model, since each term involves only a single varible

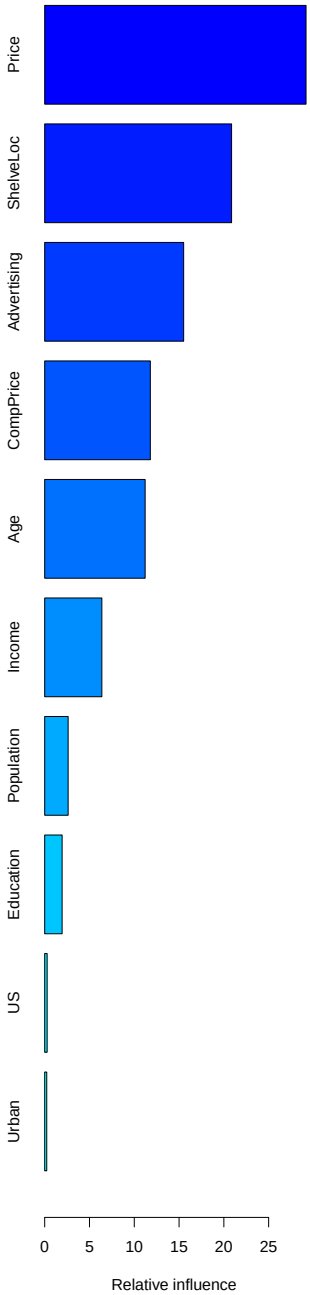
```
library(gbm)
Carseats$High <- ifelse(Carseats$High == "Yes",
    1, 0)
bm <- gbm(High ~ . - Sales, data = Carseats[train,
    ], distribution = "bernoulli", n.trees = 1000,
    shrinkage = 0.01, interaction.depth = 2)

summary(bm)
```

```
##                        var    rel.inf
## Price                Price 29.1737507
## ShelveLoc        ShelveLoc 20.8629916
## Advertising Advertising 15.5129853
## CompPrice        CompPrice 11.7916369
## Age                    Age 11.2127702
## Income              Income  6.3803898
## Population    Population  2.6168676
## Education      Education  1.9452989
## US                    US  0.2740341
## Urban              Urban  0.2292749
```

```
par(mfrow = c(1, 3))
plot(bm, i = "Price")
plot(bm, i = "ShelveLoc")
plot(bm, i = "CompPrice")
```




```
bm.pred <- predict(bm, Carseats[-train, ], n.trees = 1000,
    type = "response")
bm.pred <- ifelse(bm.pred > 0.5, "Yes", "No")
table(bm.pred, Carseats$High[-train])
```



```
##
## bm.pred  0  1
##     No  51  6
##     Yes  8 35
```