HW4

Nan Tang

5/15/2020

8.4.2

Since Boosting is a stagewise regression, adding basis function to the model will not change coefficients of previous basis.

Suppose there are k stumps choose predictor X_j, then there are k basis functions. If $Z_{j1} < Z_{j2} ... < Z_{jk}$, then basis funtions are

$$\begin{aligned}
B_{j1}(Z) &= I(Z_{j1} < X_j \le Z_{j2}) \\
B_{j2}(Z) &= I(Z_{j2} < X_j \le Z_{j3}) \\
&... \\
B_{jk}(Z) &= I(Z_{jk} < X_j) \\
B_j &= \sum_{i=1}^{k} B_{ji}
\end{aligned}$$

then for any predictor X_j, $f_j(X_j)$ can be represented as (λ is learning rate)

$$\begin{aligned}
f_j(X_j) &= \lambda[c_1 B_{j1} + c_2 B_{j2} ... + c_k B_{jk}] \\
&= \lambda \sum_{i=1}^{k} c_i B_{ji}
\end{aligned}$$

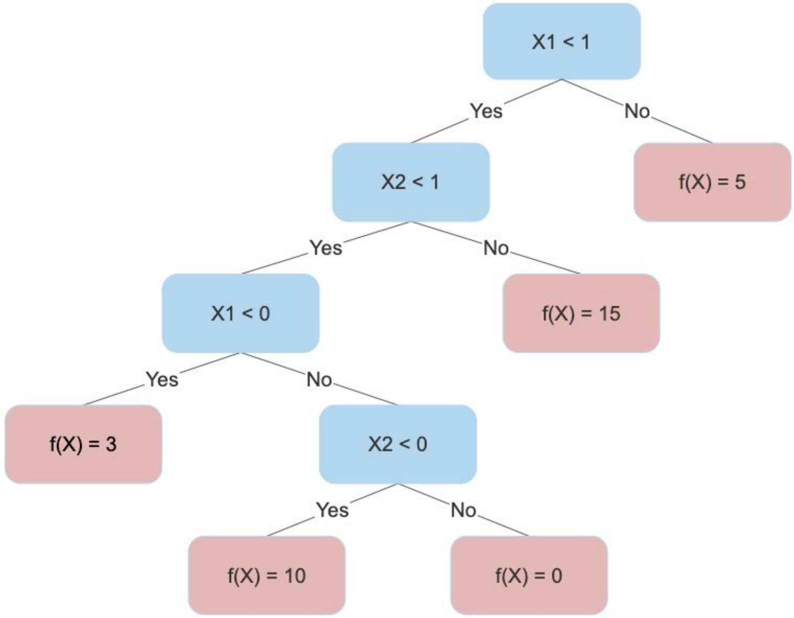
Since each function f_j depends only on single predictor X_j, the model can be represented as

$$f(X) = \sum_{j=1}^{p} f_j X_j$$

8.4.4

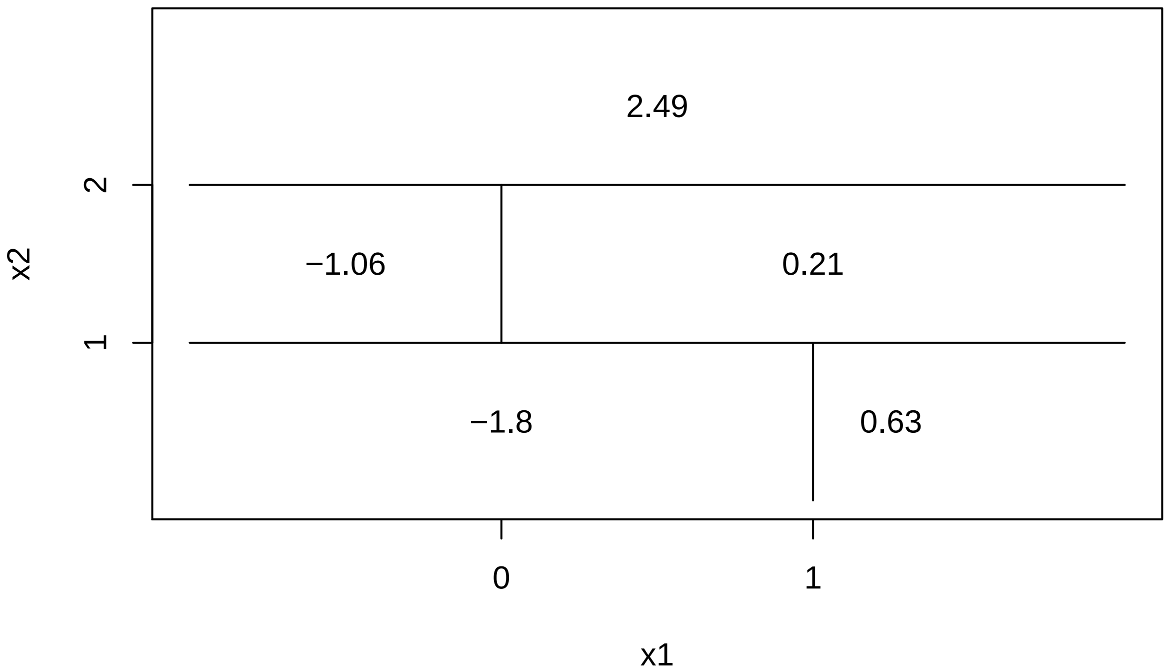
a

```r
knitr::include_graphics("/Users/nantang/Google Drive/STAT435/HW/HW4/8-4-4-a.jpg")
```



b

```
x1_lim = c(-1, 2)
x2_lim = c(0, 3)

plot(NA, NA, xlim=x1_lim, xlab='x1', ylim=x2_lim, ylab='x2',
xaxt="n", yaxt="n")

axis(side=2, at=c(1, 2))
axis(side=1, at=c(0, 1))

lines(x=x1_lim, y=c(1, 1))
lines(x=c(1, 1), y=c(x2_lim[1], 1))
lines(x=x1_lim, y=c(2, 2))
lines(x=c(0, 0), y=c(1, 2))

text(x=c(0, 1.25, -0.5, 1, 0.5), y=c(0.5, 0.5, 1.5, 1.5, 2.5),
     labels=c(-1.8, 0.63, -1.06, 0.21, 2.49))
```



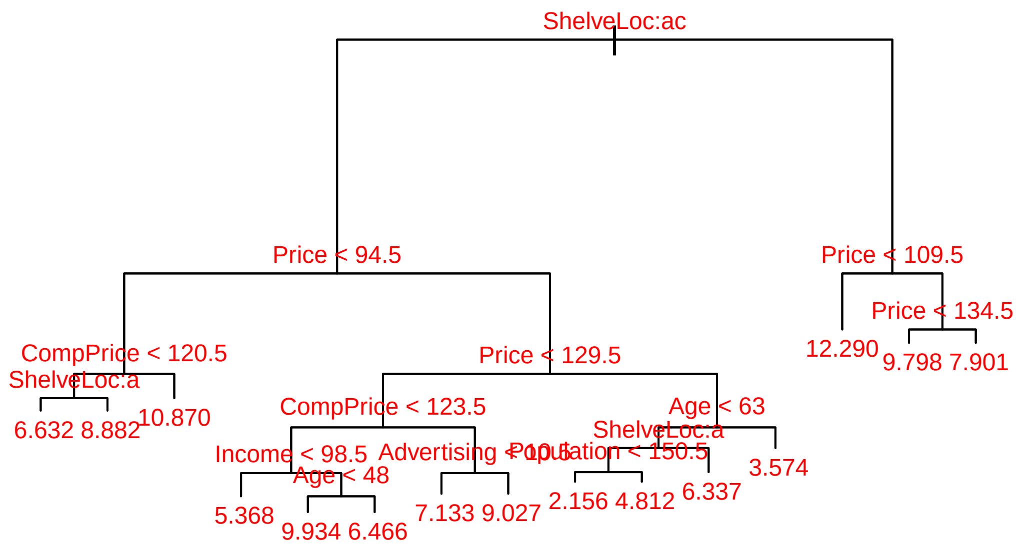
8.4.8

b

```
train_tree <- tree(formula = Sales~., data=Carseats.train)

test_pred <- predict(train_tree, Carseats.test)

test_mse <- mean((test_pred - Carseats.test$Sales)^2)
print(test_mse)
```

```
## [1] 4.965909
```

```
plot(train_tree)
text(train_tree, cex=0.75, col='red')
```



The expected value of Sales for observations with ‘Price < 94.5’, ‘CompPrice < 120.5’, and ‘ShelveLoc = Bad’ is 6.632.

The expected value of Sales for observations with ‘Price < 94.5’, ‘CompPrice < 120.5’, and ‘ShelveLoc = Medium’ is 8.882.

The expected value of Sales for observations with ‘Price < 94.5’, ‘CompPrice >= 120.5’, and ‘ShelveLoc = Medium or Bad’ is 10.870.

The expected value of Sales for observations with ‘94.5 <= Price < 129.5’, ‘CompPrice < 123.5’, ‘Income < 98.5’, and ‘ShelveLoc = Medium or Bad’ is 5.368.

The expected value of Sales for observations with ‘94.5 <= Price < 129.5’, ‘CompPrice < 123.5’, ‘Income >= 98.5’, ‘Age < 48’, and ‘ShelveLoc = Medium or Bad’ is 9.934.

The expected value of Sales for observations with ‘94.5 <= Price < 129.5’, ‘CompPrice < 123.5’, ‘Income >= 98.5’, ‘Age >= 48’, and ‘ShelveLoc = Medium or Bad’ is 6.466.

The expected value of Sales for observations with ‘94.5 <= Price < 129.5’, ‘CompPrice >= 123.5’, ‘Advertising < 10.5’, and ‘ShelveLoc = Medium or Bad’ is 7.133.

The expected value of Sales for observations with ‘94.5 <= Price < 129.5’, ‘CompPrice >= 123.5’, ‘Advertising >= 10.5’, and ‘ShelveLoc = Medium or Bad’ is 9.027.

The expected value of Sales for observations with ‘Price >= 129.5’, ‘Age < 63’, ‘Population < 150.5’, and ‘ShelveLoc = Bad’ is 2.156.

The expected value of Sales for observations with ‘Price >= 129.5’, ‘Age < 63’, ‘Population >= 150.5’, and ‘ShelveLoc = Bad’ is 4.812.

The expected value of Sales for observations with ‘Price >= 129.5’, ‘Age < 63’, and ‘ShelveLoc = Medium’ is 6.337.

The expected value of Sales for observations with ‘Price >= 129.5’, ‘Age >= 63’, and ‘ShelveLoc = Medium or Bad’ is 3.574.

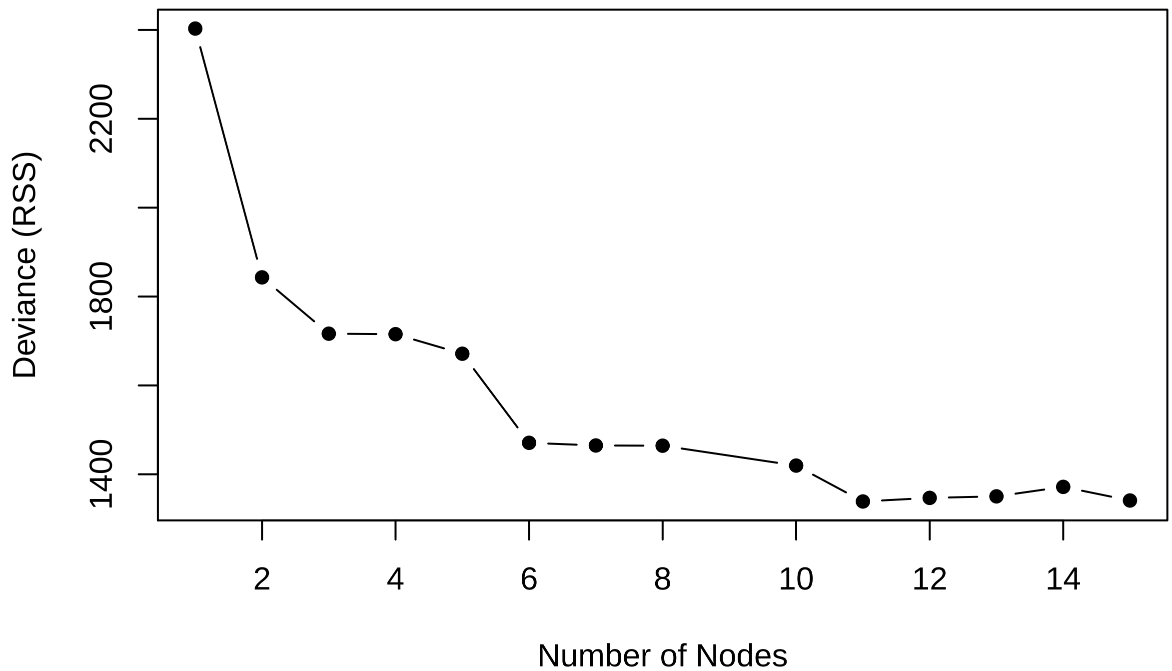
The expected value of Sales for observations with ‘ShelveLoc = Good’, and ‘Price < 109.5’ is 12.29.

The expected value of Sales for observations with ‘ShelveLoc = Good’, and ‘109.5 <= Price < 134.5’ is 9.798.

The expected value of Sales for observations with ‘ShelveLoc = Good’, and ‘Price >= 134.5’ is 7.901.

c

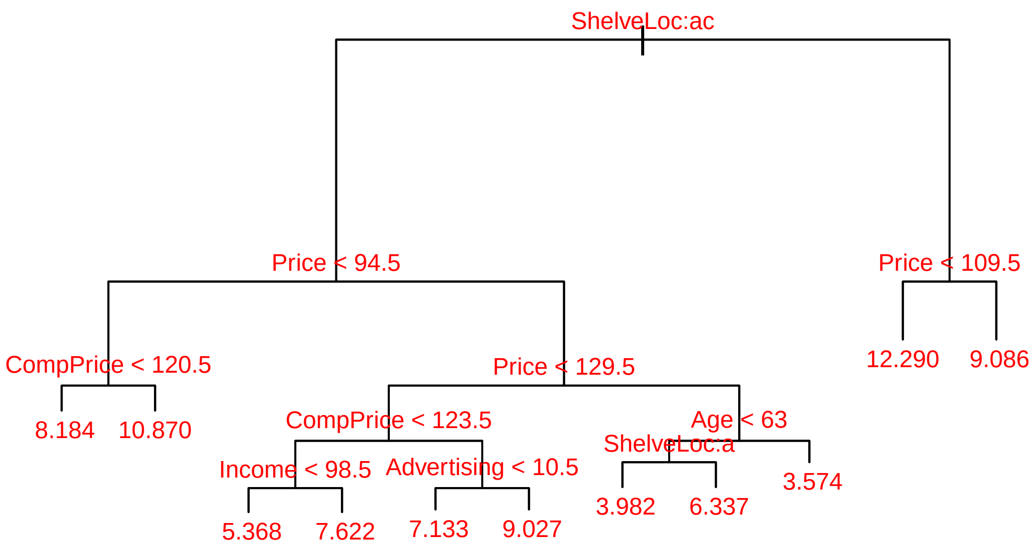
```r
set.seed(123)

## base on deviance
train_cv <- cv.tree(train_tree)
plot(rev(train_cv$size), rev(train_cv$dev), type='b', pch=16,
     xlab='Number of Nodes', ylab='Deviance (RSS)')
```




```r
best_node <- train_cv$size[which(train_cv$dev == min(train_cv$dev))]
print(best_node)
```

```
## [1] 11
```

```r
prune_tree <- prune.tree(train_tree, best=best_node)

plot(prune_tree)
text(prune_tree, cex=0.75, col='red')
```




```
prune_pred <- predict(prune_tree, Carseats.test)
prune_mse <- mean((prune_pred - Carseats.test$Sales)^2)
print(prune_mse)
```

```
## [1] 5.307632
```

In this case, optimized level of complexity chosen by cross-validation failed to improve test mse.

d

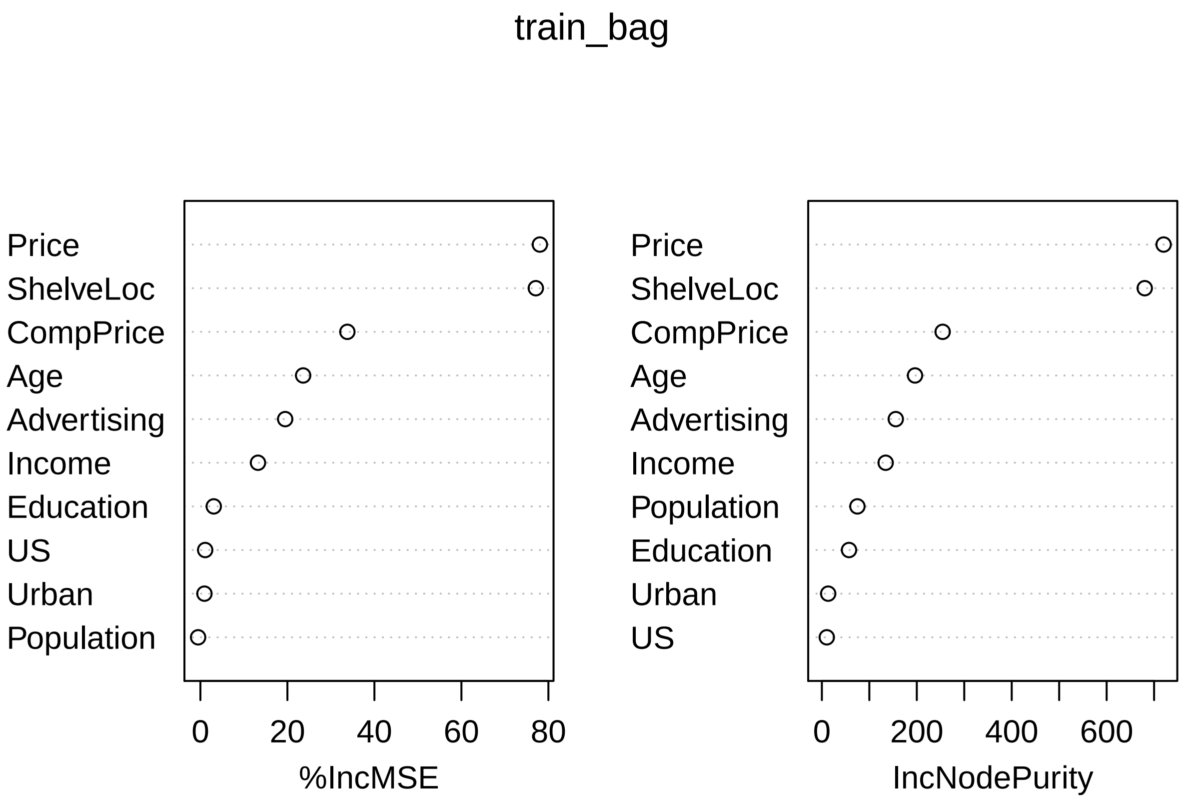
```
set.seed(123)

D <- ncol(Carseats.train) - 1

train_bag <- randomForest(formula=Sales~., data=Carseats.train, mtry=D, importance=TRUE)
test_pred <- predict(train_bag, Carseats.test)

test_mse <- mean((test_pred - Carseats.test$Sales)^2)
print(test_mse)
```

```
## [1] 2.62921
```

```
importance(train_bag)
```

```
##                %IncMSE IncNodePurity
## CompPrice   33.7824819     254.51264
## Income      13.2326243     134.39193
## Advertising 19.4864506     155.67375
## Population  -0.5286856      74.85775
## Price       78.0379499     720.11336
## ShelveLoc   77.1062401     680.28547
## Age         23.6156429     196.26516
## Education    3.0659529      57.20060
## Urban        0.9235326      13.30628
## US           1.0935985      10.35043
```

```r
varImpPlot(train_bag)
```



Predictors 'ShelveLoc' and 'Price' are most important in decreasing impurity of splits and training RSS.

Test MSE by bagging procedure is less than MSE of single decision tree.

e

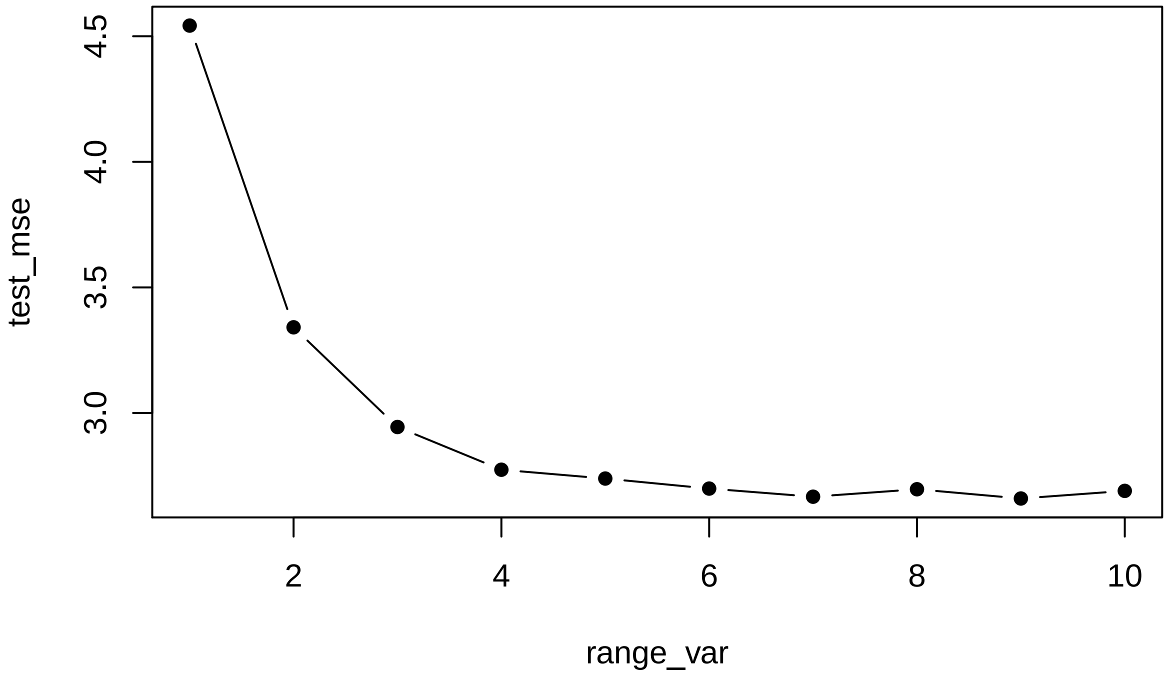
```r
set.seed(123)

## use m = D/3 \approx 3 as number of predictors in each tree
train_rf <- randomForest(formula=Sales~., data=Carseats.train, mtry=3, importance=TRUE)
test_pred <- predict(train_rf, Carseats.test)

test_mse <- mean((test_pred - Carseats.test$Sales)^2)
print(test_mse)
```

```
## [1] 2.960136
```

Following the rule that number of predictors applied in each tree equals (total/3) ≈ 3 in regression tree, we obtained test_mse approximately equal to 3.

```r
set.seed(123)

range_var <- 1:D

test_mse <- numeric(length(range_var))

for(i in 1:length(range_var)) {
```

```r
  train_rf <- randomForest(formula=Sales~., data=Carseats.train, mtry=range_var[i], importance=TRUE)
  test_pred <- predict(train_rf, Carseats.test)
  test_mse[i] <- mean((test_pred - Carseats.test$Sales)^2)
}

plot(range_var, test_mse, type='b', pch=16)
```




```r
best_m <- range_var[which(test_mse == min(test_mse))]
print(best_m)
```

```
## [1] 9
```

It turns out 9 predictors in each tree optimized test MSE.

```r
set.seed(123)

## use m = D/3 \approx 3 as number of predictors in each tree
train_rf <- randomForest(formula=Sales~., data=Carseats.train, mtry=best_m, importance=TRUE)
test_pred <- predict(train_rf, Carseats.test)

test_mse <- mean((test_pred - Carseats.test$Sales)^2)
print(test_mse)
```

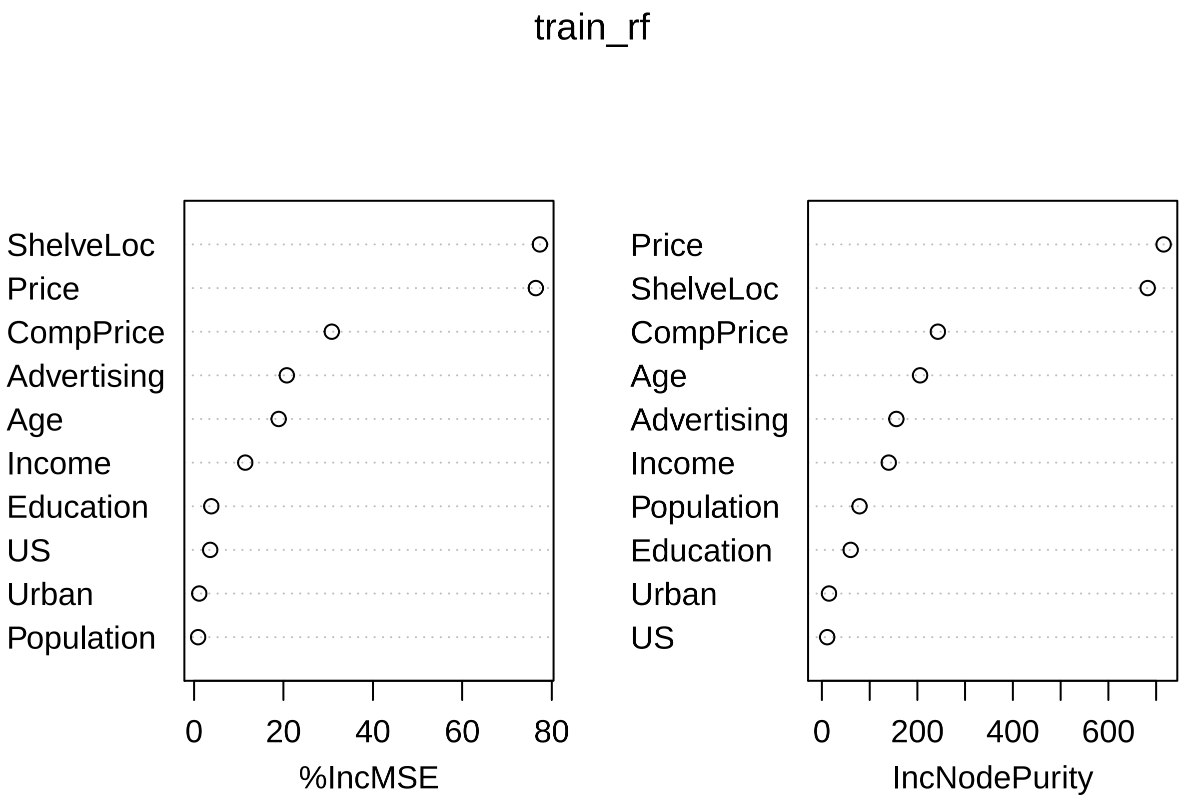
```
## [1] 2.64404
```

```r
importance(train_rf)
```

```
##                %IncMSE IncNodePurity
## CompPrice   30.8081431     242.96069
## Income      11.4588632     139.99843
## Advertising 20.7462180     155.95451
## Population   0.9075274      78.83739
## Price       76.4509153     715.71083
## ShelveLoc   77.3612266     682.30356
```

```
## Age          18.9296370     205.72320
## Education     3.8607126      60.20539
## Urban         1.1627837      15.01071
## US            3.6060744      11.19286
```

```
varImpPlot(train_rf)
```



For random forest with 9 predictors considered in each split, 'Price' and 'ShelveLoc' are most important ones in predicting expected value of Sales.