Chapter 8 Applied Exercises

To start with a clean session, we first remove all packages except the base R packages (we will load anything needed later) and all variables.

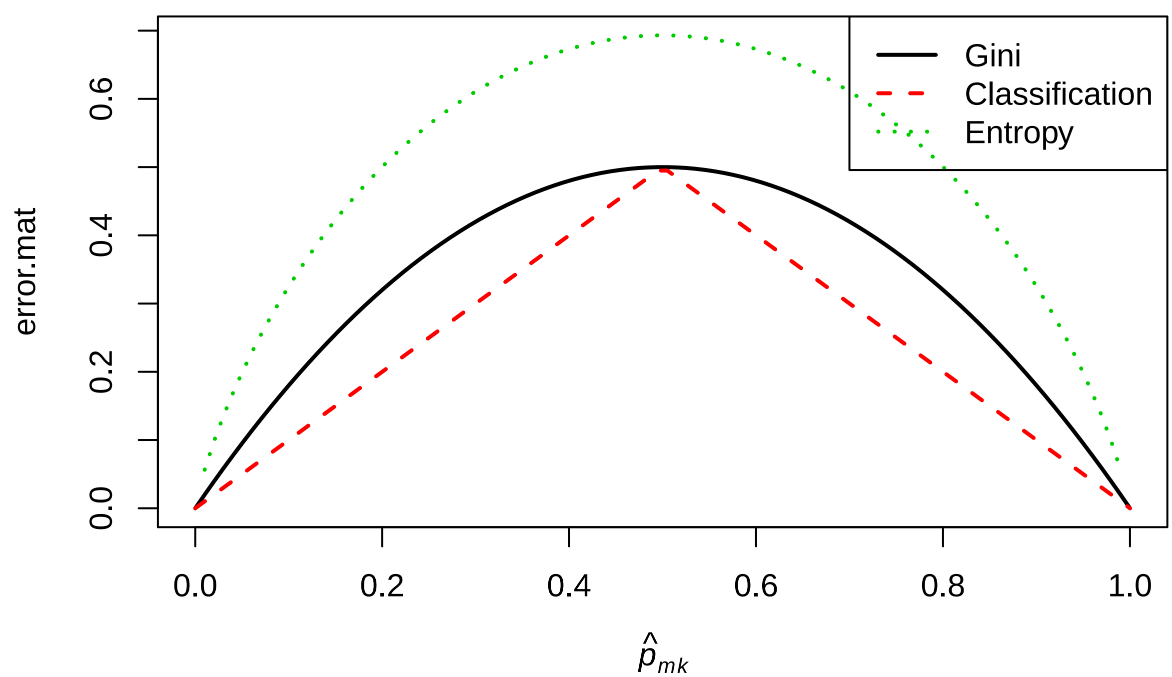
```
rm(list = ls())
pkgs = names(sessionInfo()$otherPkgs)
if (!is.null(pkgs)) {
  detach_list <- paste0("package:", pkgs)
  lapply(detach_list, FUN = detach, character.only = TRUE)
}
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
##
## [[9]]
## NULL
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
##
## [[13]]
## NULL
##
## [[14]]
## NULL
```

```
##
## [[15]]
## NULL
##
## [[16]]
## NULL
```

Exercise 3

```r
p = seq(0,1, length = 100)
# Gini Index
gini= 2*p*(1-p)
err = apply(cbind(1-p, p), FUN = min, MARGIN = 1)
entropy = -p*log(p) - (1-p)*log(1-p)
error.mat = cbind(gini, err, entropy)
matplot(p, error.mat, xlab = expression(italic(hat(p)[mk])), type = "l", lwd = 2, col = 1:3, lty = 1:3)
legend("topright", legend = c("Gini", "Classification", "Entropy"), lwd = 2, col = 1:3, lty = 1:3)
```



To see how to write mathematical expressions on a plot, type `?plotmath`. Pairwise minimum command `pmin` could be used above instead of the `apply()` function.

Exercise 7

To see not only how changes in the number of trees and m affect the test error, but also how they interact, we depict them on the same plot. We find the test MSE as a function of the number of trees for values $m = p, p/2, \sqrt{p}$.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(MASS)
# define test and train
set.seed(1)
train = sample(1:nrow(Boston), size = nrow(Boston)/2)
test = -train
test.y = Boston$medv[test]
# Grids for ntree and mtry
(ntree_seq = seq(1, 1000, by = 20))
```

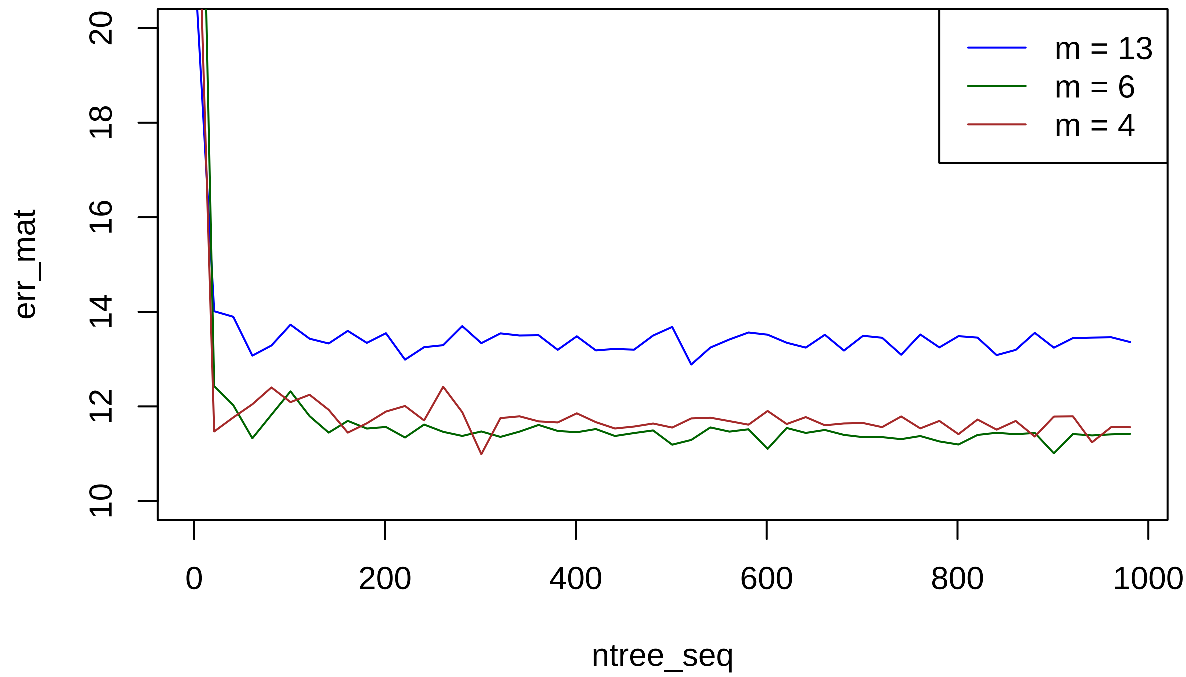
```
##  [1]   1  21  41  61  81 101 121 141 161 181 201 221 241 261 281 301 321
## [18] 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661
## [35] 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981
```

```
(mtry_seq = c(13, round(13/2), round(sqrt(13))))
```

```
## [1] 13  6  4
```

```
err_mat = matrix(NA, nrow = length(ntree_seq), ncol = length(mtry_seq),
                 dimnames = list(NULL, NULL))
# error estimates for the grid
for (i_tree in 1:length(ntree_seq)) {
  for (j_mtry in 1:length(mtry_seq)) {
    rf_boston = randomForest(medv ~ ., data = Boston, ntree = ntree_seq[i_tree],
                             mtry = mtry_seq[j_mtry], subset = train)
    rf_pred = predict(rf_boston, newdata = Boston[test, ])
    err_mat[i_tree, j_mtry] = mean((rf_pred - test.y)^2)
  }
}
```

```
matplot(ntree_seq, err_mat, type = "l", lty = 1, col = c("blue", "darkgreen", "brown"), lwd = 1, ylim =
legend("topright", legend = paste0("m = ", mtry_seq), col = c("blue", "darkgreen", "brown"), lwd = 1)
```



Using 400 trees is sufficient to give good performance and $m \simeq p/2 = 6$ leads to the best test MSE. Random forest outperforms bagging in this example.

The 3D plot shows that a very small m would result in large error, and the error falls as we increase m. This plot is, however, difficult to interpret. So we make an attempt toward drawing 2D plots which could reflect the interaction between the two variable of interest.
In order to use the pacakge `ggplot2`, we need to tidy the data first, i.e. make each row of the data represent one point.
Suppose we have a grid $x = \{x_i\}_{i=1}^n$ on the X-axis and $y = \{y_i\}_{i=1}^q$ on the y-axis and a matrix $Z_{n\times q} = [z_{ij}]_{ij}$ where $z_{ij} = f(x_i, y_j)$. The goal is to represent each point as a single row, i.e. make a one-to-one correspondence between the grids, i.e. between vectors x and y, and also between the grids and the matrix Z. To achieve this, we create a matrix $X_{n\times q}$ out of vector x by copying x to each column of X. We also create Y by copying the vector y to each row of the matrix $Y_{n\times q}$.
Now let x be the number of splits and y equal to m, the number of predictors considered in each split. We have

```
X = matrix(ntree_seq, nrow = length(ntree_seq), ncol = length(mtry_seq))
Y = matrix(mtry_seq, nrow = length(ntree_seq), ncol = length(mtry_seq), byrow = TRUE)
head(X)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [2,]    3    3    3    3    3    3    3    3    3     3     3     3     3
## [3,]    5    5    5    5    5    5    5    5    5     5     5     5     5
## [4,]    7    7    7    7    7    7    7    7    7     7     7     7     7
## [5,]    9    9    9    9    9    9    9    9    9     9     9     9     9
## [6,]   11   11   11   11   11   11   11   11   11    11    11    11    11
```

```r
head(Y)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [2,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [3,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [4,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [5,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [6,]    1    2    3    4    5    6    7    8    9    10    11    12    13
```

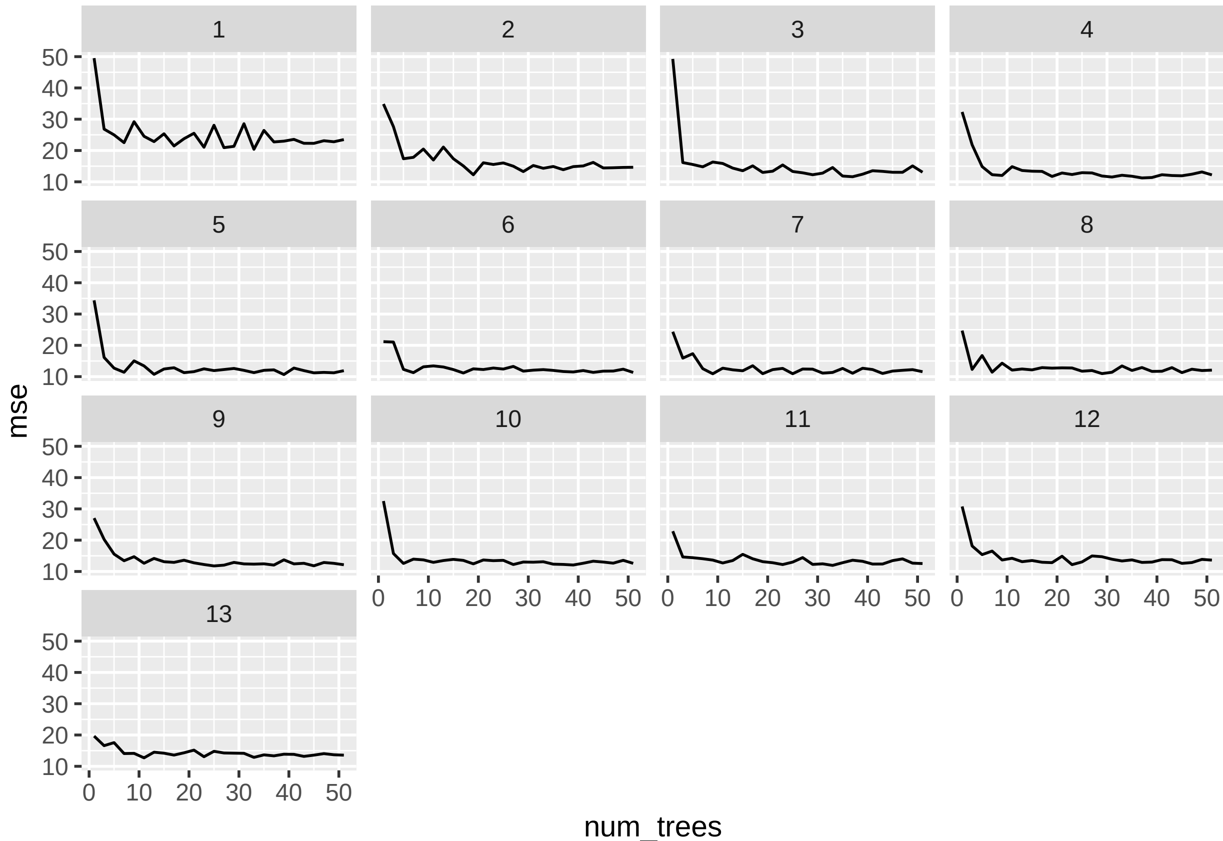
Note the use of `byrow = TRUE`. Now that all elements of matrices `X`, `Y` and `err_mat` correspond, we generate a data frame by converting all these matrices to columns:

```r
df_tree = data.frame(num_trees = c(X), num_vars = c(Y), mse = c(err_mat))
head(df)
```

```
##
## 1 function (x, df1, df2, ncp, log = FALSE)
## 2 {
## 3     if (missing(ncp))
## 4         .Call(C_df, x, df1, df2, log)
## 5     else .Call(C_dnf, x, df1, df2, ncp, log)
## 6 }
```

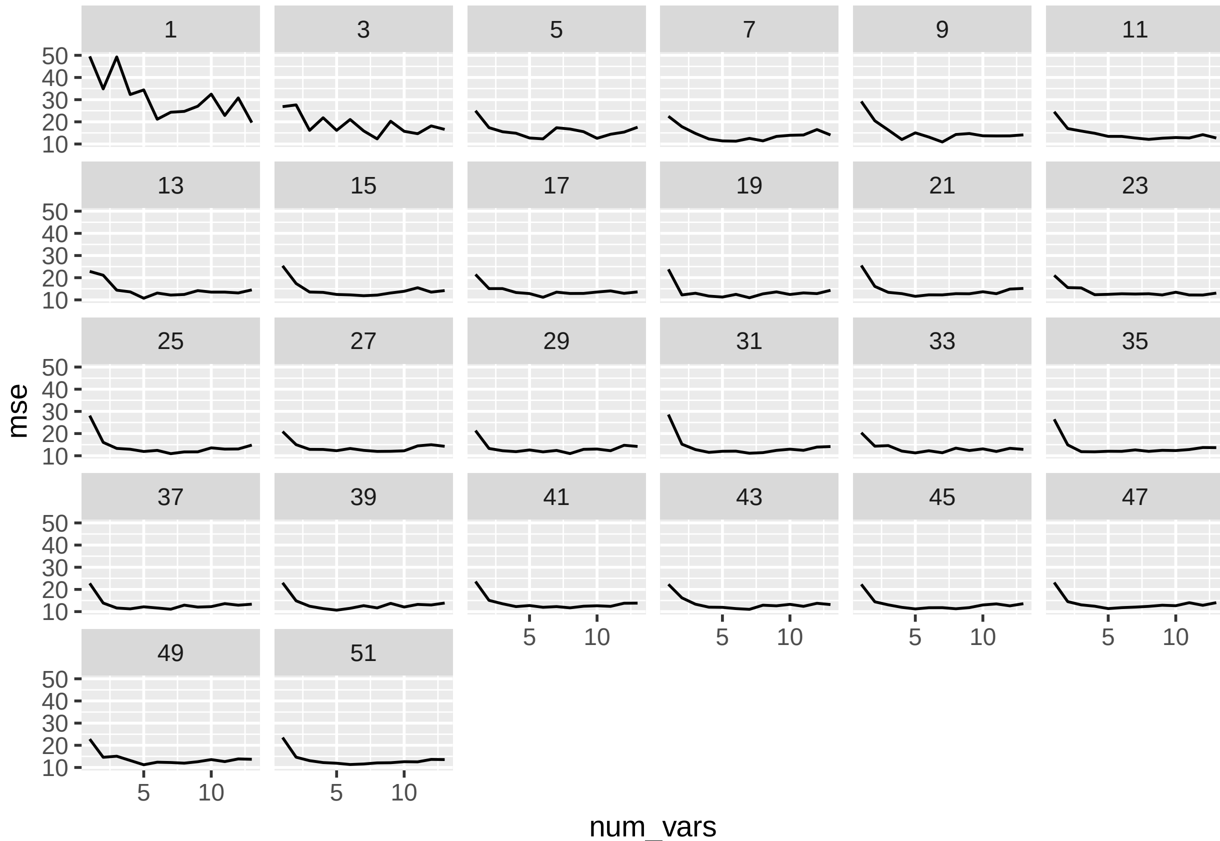
We can use the data frame `df_tree` in `ggplot2` to draw different plots.

```r
library(ggplot2)
ggplot(data = df_tree) +
  geom_line(mapping = aes(x = num_trees, y = mse)) +
  facet_wrap(~num_vars)
```



Using the number of variables as the facet, we could see how the number of trees affect the error estimate. By increasing the number of trees to 5, we see a considerable improvement in accuracy for different levels of m, and the improvement slows down for larger number of trees.
We might be tempted to state that increasing the number of variables does not have much effect on reducing the error for m greater than 4, implying that $m = 4$ might be a good choice. However, note that this is not the best plot for making this conclusion. The y-axis scale is too large to be able to distinguish the differences between the error values for large enough values of `num_trees` and `num_vars`. To see how the plot above might hide such information, consider `num_trees` as the facet:

```
ggplot(data = df_tree) +
  geom_line(mapping = aes(x = num_vars, y = mse)) +
  facet_wrap(~num_trees)
```



Using the number of trees as the facet, we see that the error has a downward trend and stabilizes at about 5 variables, given the values of `num_trees`. We can further adjust the scale on the y-axis to see the trend. We could also explore how changing the random seed might affect our conclusion, or investigate the effect of increasing bootstrap samples, but we will not purue either of these issues here.

–>

Exercise 8

Part 8.a)

We will predict `Sales` in the `Carseats` data, treating it as a quantitative variable.

```
library(ISLR)
library(tree)
# split the data
set.seed(1)
train = sample(1:nrow(Carseats), size = nrow(Carseats)/2)
test = -train
test.y = Carseats$Sales[test]
```

Part 8.b)

```r
# fit regression tree
tree.carseats = tree(Sales ~ ., data = Carseats, subset = train)
tree.carseats
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 200 1526.000  7.335
##    2) ShelveLoc: Bad,Medium 164 1084.000  6.815
##      4) Price < 120.5 103  541.000  7.776
##        8) Age < 50.5 39  133.100  9.278
##         16) Price < 104.5 25   83.050  9.808
##           32) ShelveLoc: Bad 7   22.340  8.284 *
##           33) ShelveLoc: Medium 18   38.140 10.400 *
##         17) Price > 104.5 14   30.510  8.333
##           34) Advertising < 3.5 5    6.584  6.922 *
##           35) Advertising > 3.5 9    8.444  9.117 *
##        9) Age > 50.5 64  266.200  6.860
##         18) Price < 92 17   67.560  8.628
##           36) Income < 85 9   23.830  7.512 *
##           37) Income > 85 8   19.940  9.882 *
##         19) Price > 92 47  126.300  6.221
##           38) ShelveLoc: Bad 16   29.890  5.059 *
##           39) ShelveLoc: Medium 31   63.660  6.820
##             78) CompPrice < 107 5    4.229  5.206 *
##             79) CompPrice > 107 26   43.900  7.131 *
##      5) Price > 120.5 61  287.000  5.192
##       10) Age < 66.5 49  181.400  5.654
##         20) CompPrice < 148 37  129.600  5.208
##           40) Advertising < 10.5 25   75.180  4.627 *
##           41) Advertising > 10.5 12   28.370  6.418 *
##         21) CompPrice > 148 12   21.680  7.031 *
##       11) Age > 66.5 12   52.410  3.303
##         22) Price < 132 5   16.300  4.780 *
##         23) Price > 132 7   17.410  2.249 *
##    3) ShelveLoc: Good 36  196.000  9.706
##      6) Price < 113 10   19.960 12.080 *
##      7) Price > 113 26   97.810  8.792
##       14) Price < 142.5 20   54.100  9.337
##         28) CompPrice < 133 15   24.220  8.778 *
##         29) CompPrice > 133 5   11.130 11.010 *
##       15) Price > 142.5 6   17.930  6.973 *
```

What if we use Gini index for growing the tree?

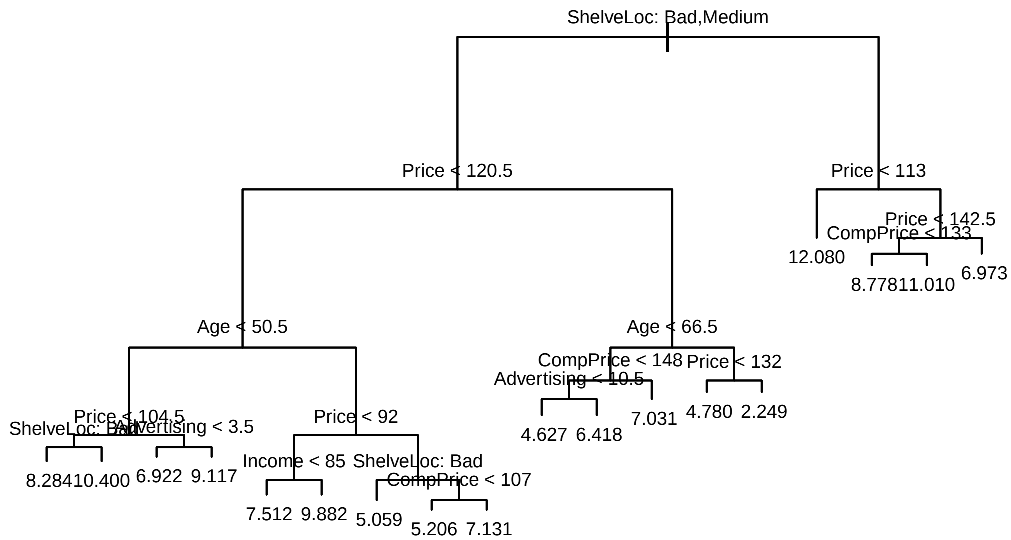
```r
tree.carseats_gini = tree(Sales ~ ., data = Carseats, subset = train, split = "gini")
tree.carseats_gini
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 200 1526 7.335 *
```

It leads to no partitioning! Hence, it seems best to use other packages such as `rpart` for splitting according to the Gini index. It is also not clear to me the default splitting accroding to deviance is equivalent to cross-entropy. A little algebra shows that deviance contribution of each region R_m should be normalized by

n_m to become equivalent to cross-entropy.
As a result of these points, although the package `tree` could be a strating point, it is not my choice for final results in a regression tree. However, we continue to use the package `tree` and the default for its argument `split` which is `deviance`.

```r
# plot the tree
plot(tree.carseats)
text(tree.carseats, pretty = FALSE, cex = 0.6)
```




```r
#
```

Similar to the classification tree considered in the chapter's lab, shelving location seems to be the most important variable. The first branch differentiates the `Good` locations from `Medium` and `Bad` locations. Price also appears to be important, as many branches distinguish levels of price.

Advertising, which is potentially an important decision variable, is correlated with sales on a medium range of prices for `Good` shelving locations. It is expected that advertising would affect segments of the market that face less price competition by differentiating their products. Hence, as expected, advertising is not likely to affect the low-end products which are likely to compete through prices and lack much brand recognition. On the other hand, the very high-end products are likely to be produced in a smaller scale, which might not make advertising on a large scale worthwhile.

```r
# predict for test observations
tree_preds = predict(tree.carseats, newdata = Carseats[test, ])
# test MSE:
mse_tree = mean((tree_preds - test.y)^2)
mse_tree
```

```
## [1] 4.148897
```

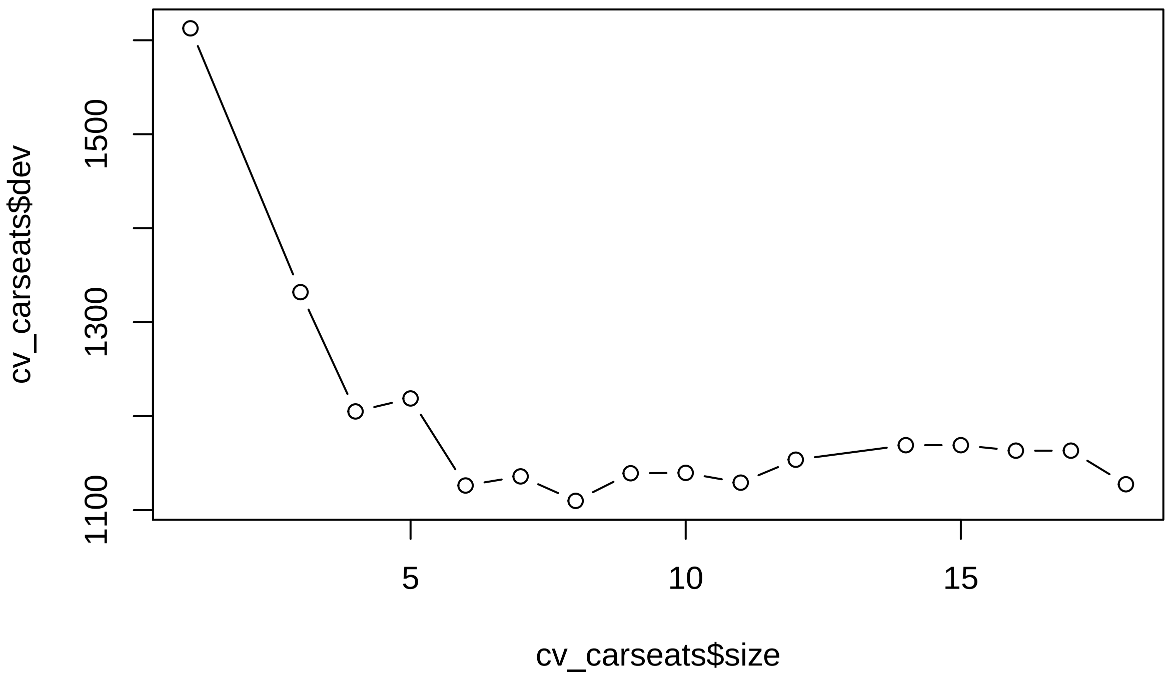
Part 8.c)

Now we prune the tree to see whether we can improve prediction accuracy.

```
set.seed(1)
cv_carseats = cv.tree(tree.carseats)
cv_carseats
```

```
## $size
##  [1] 18 17 16 15 14 12 11 10  9  8  7  6  5  4  3  1
##
## $dev
##  [1] 1127.510 1163.270 1163.270 1169.097 1169.097 1153.636 1129.237
##  [8] 1139.639 1139.232 1109.839 1135.897 1126.203 1218.821 1205.007
## [15] 1331.927 1612.664
##
## $k
##  [1]      -Inf  15.48181  15.53599  18.69038  18.74886  21.05038  23.79480
##  [8]  25.78579  26.01210  30.10435  32.74801  53.28569  72.33061  78.19599
## [15] 141.73781 251.22901
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv_carseats$size, cv_carseats$dev, type = "b")
```



The cross-validation MSE estimates is minimized for 8 splits.

```
# find the pruned tree
prune_carseats = prune.tree(tree.carseats, best = 8)
# predict for the pruned tree
prune_preds = predict(prune_carseats, newdata = Carseats[test, ])
# test mse for the pruned tree
mse_prune_carseats = mean((prune_preds - test.y)^2)
mse_prune_carseats
```

```
## [1] 5.09085
```

It results in larger test MSE, so seems not to improve accuracy (for this specific test set).

Exercise 8.d)

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(1)
bag_carseats = randomForest(Sales ~ ., data = Carseats, subset = train,
                            mtry = ncol(Carseats)-1, ntree = 500, importance = TRUE)
preds_bag_carseats = predict(bag_carseats, newdata = Carseats[test, ])
mse_bag_carseats = mean((preds_bag_carseats - test.y)^2)
mse_bag_carseats
```

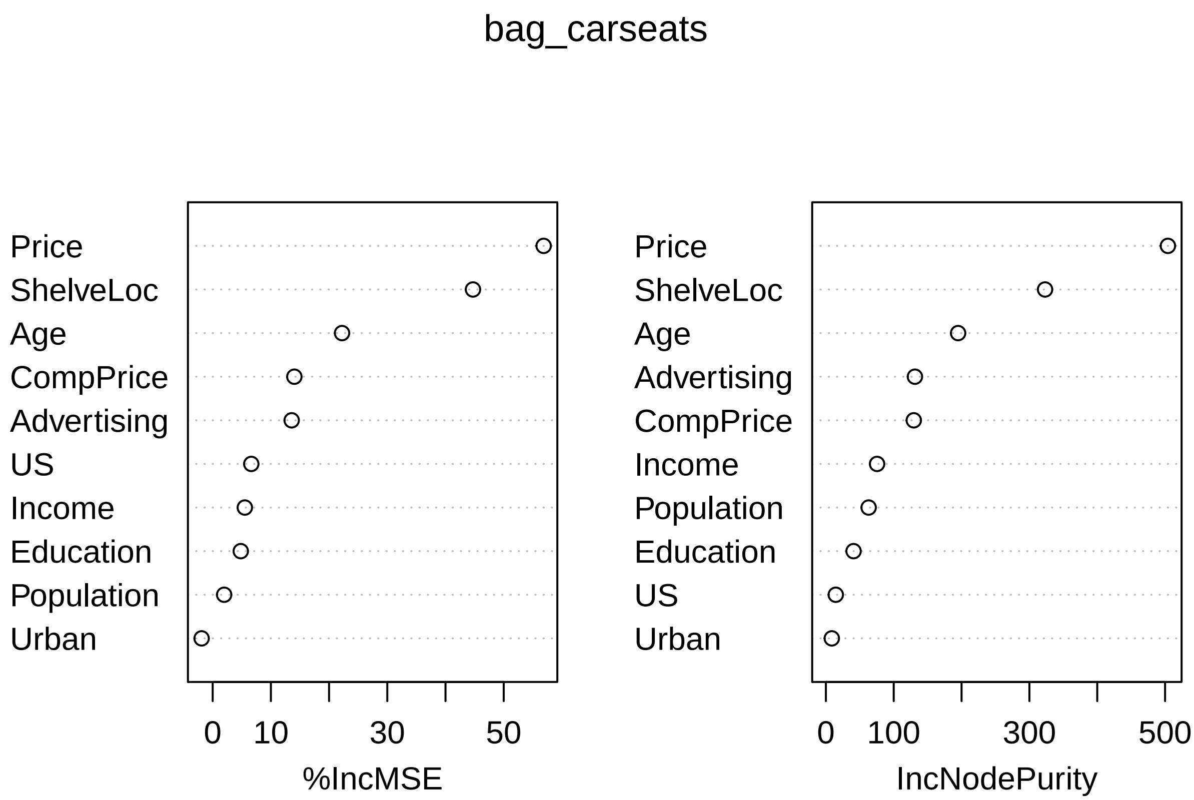
```
## [1] 2.554292
```

Using bagging would improve the prediciton accuracy.
Note that we set `mtry` equal to number of all the predictors, which is supposed to give us the bagging estimator, which is a special case of random forest estimator. However, it appears that doing so would not exactly result in a bagging estimator, as introduced in the textbook. We would expect the baggin estimator to be certain, since there is no randomness when all predictors are considered in each split. However, it appears that the algorithm for `randomForest` generates other sources of randomness. That is why we have set the seed in the code chunk above.
If we do not set the random seed, we would get different results each time we we use `randomForest()` even for maximum value of `mtry` and `ntree` being equal to one.
Now we determine which variables are most important:

```r
importance(bag_carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice   14.032030    129.568747
## Income       5.523038     75.448682
## Advertising 13.571285    131.246840
## Population   1.968853     63.042648
## Price       56.863812    504.158108
## ShelveLoc   44.720455    323.055042
## Age         22.225468    194.915976
## Education    4.823966     40.810991
## Urban       -1.902185      8.746566
## US           6.632887     14.599565
```

```r
varImpPlot(bag_carseats)
```



The variables Price and ShelveLocappear to be the most important variables according to both measures, one based on contribution of the variable to the out-of-bag estimate of the test error (on the LHS) and the other based on contribution of the variable to the total node impurity in the training data (on the RHS). The two importance measure generally yield similar results in terms of relative importance of the variables in determining sales. An exception is the variableUS' which is more important according to the OOB estimates.

Part 8.e)

```
set.seed(1)
rf_carseats <- randomForest(Sales ~ ., data = Carseats, subset = train,
                            ntree = 500, importance = TRUE)
preds_rf_carseats <- predict(rf_carseats, newdata = Carseats[test, ])
mse_rf_carseats <- mean((preds_rf_carseats - test.y)^2)
mse_rf_carseats
```

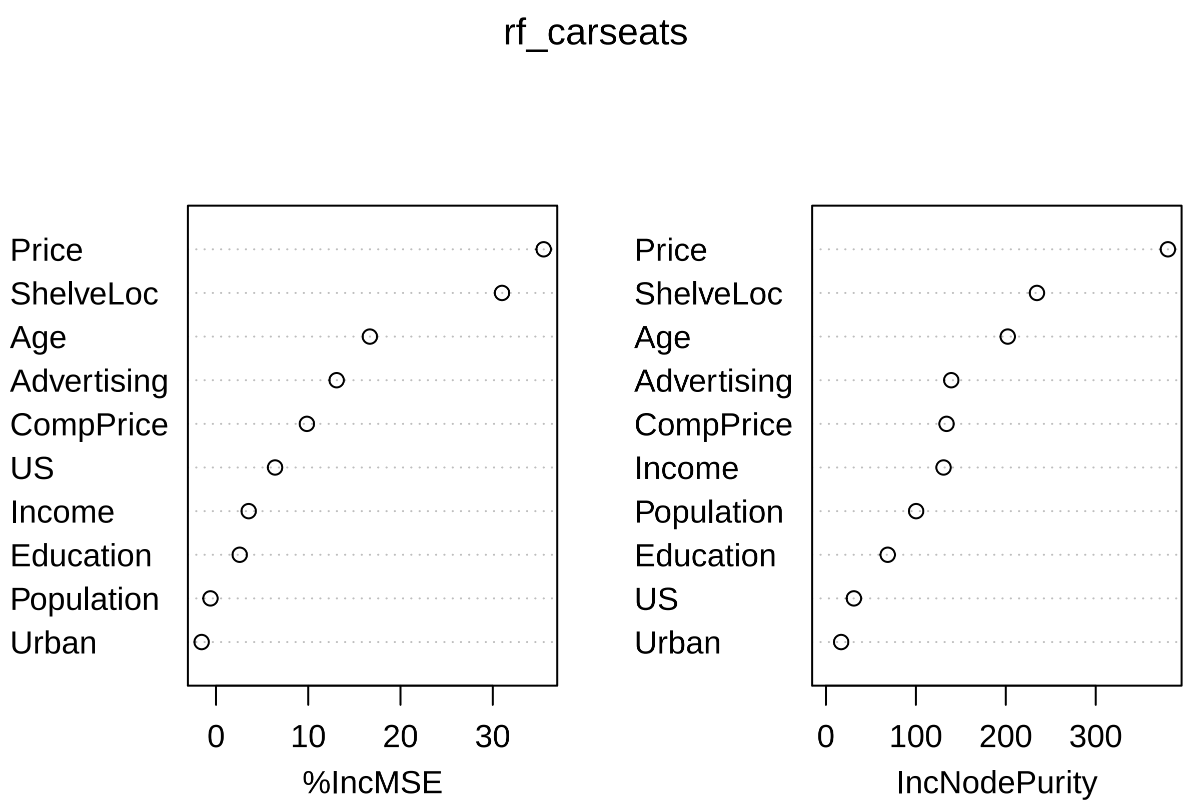
```
## [1] 3.30763
```

```
importance(rf_carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice     9.849043     134.17665
## Income        3.534622     130.84360
## Advertising 13.075334     139.40128
## Population  -0.612195     100.34668
## Price       35.530402     380.27956
```

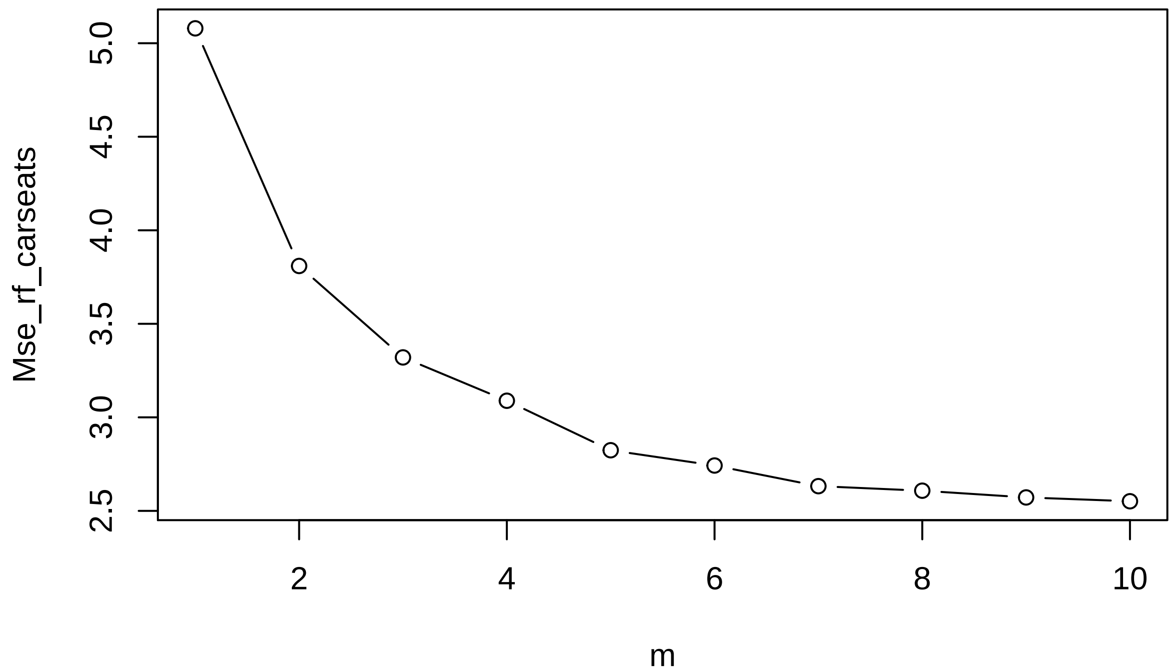
```
## ShelveLoc    31.015873     234.62966
## Age          16.680174     202.18673
## Education     2.563741      68.75977
## Urban        -1.569224      16.99182
## US            6.400241      31.12594
```

```r
varImpPlot(rf_carseats)
```



The random forest estimator has larger estimate for test MSE, but leads to similar relative importance for variables. The optimal value of `m` would depend on the bias-variance trade-off: the lower the number of variables considered in each split, the higher the variance, but the lower the correlation betweeen variables and as a result, the variance of the estimator.
Below, we see how the prediction accuracy changes with changes in `m`:

```r
set.seed(1)
Mse_rf_carseats <- rep(NA, length = 10)
for (i in 1:10) {
  rf_temp <- randomForest(Sales ~ ., data = Carseats, subset = train,
                          mtry = i, ntree = 500, importance = TRUE)
  preds_temp = predict(rf_temp, newdata = Carseats[test, ])
  Mse_rf_carseats[i] = mean((preds_temp - test.y)^2)
}
plot(1:10, Mse_rf_carseats, xlab = "m", type = "b")
```



The bagging estimator appears to have the lowest (estimate of) test MSE among different random forest estimators. Hence, the decrease is variance due to reducing `m` is not enough to offset the larger bias resulted by considering fewer variables in each split.
This might be related to the idea of why we would use small values of `m` when we have many highly correlated variables (mentioned in the textbook). On one hand, we find `m` to be the largest here and on the other hand, we have a small number of observations and variables, and the variables are not highly correlated

Exercise 9

```
rm(list = ls())
library(ISLR)
# knowing the data
dim(OJ)  # 1070*18
```

```
## [1] 1070   18
```

```
sum(is.na(OJ))
```

```
## [1] 0
```

```
summary(OJ)  # Purchase and Store7 factor variables
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH         PriceMM     
##  CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690  
##  MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990  
##           Median :257.0   Median :3.00   Median :1.860   Median :2.090  
##           Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085  
```

```
##           3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180
##           Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290
##      DiscCH            DiscMM         SpecialCH        SpecialMM
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH           SalePriceMM     SalePriceCH      PriceDiff
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
##  Store7      PctDiscMM        PctDiscCH       ListPriceDiff
##  No :714   Min.   :0.0000   Min.   :0.00000   Min.   :0.000
##  Yes:356   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140
##            Median :0.0000   Median :0.00000   Median :0.240
##            Mean   :0.0593   Mean   :0.02731   Mean   :0.218
##            3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300
##            Max.   :0.4020   Max.   :0.25269   Max.   :0.440
##      STORE
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :2.000
##  Mean   :1.631
##  3rd Qu.:3.000
##  Max.   :4.000
```

```r
# knowing the response
contrasts(OJ$Purchase)  # MM is 1 and CH is 0; MM
```

```
##    MM
## CH  0
## MM  1
```

Part 9.a)

```r
train = sample(1:nrow(OJ), size = 800)
test = -train
test_purchase = OJ$Purchase[test]
```

Part 9.b)

```r
library(tree)
tree_oj <- tree(Purchase ~ ., data = OJ, subset = train)
summary(tree_oj)
```

```
##
## Classification tree:
```

```
## tree(formula = Purchase ~ ., data = OJ, subset = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"        "SalePriceMM"    "SpecialCH"      "WeekofPurchase"
## [5] "ListPriceDiff"  "PctDiscMM"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7141 = 564.8 / 791
## Misclassification error rate: 0.1512 = 121 / 800
```

Deviance is a measure of fit which is equivalent to RSS for cases such as least squares. The smaller the deviance, the better the fit (to the training data). The training error rate is 0.15125 and the tree has 9 terminal nodes.

Part 9.c)

Here is a detailed text summary of the table:

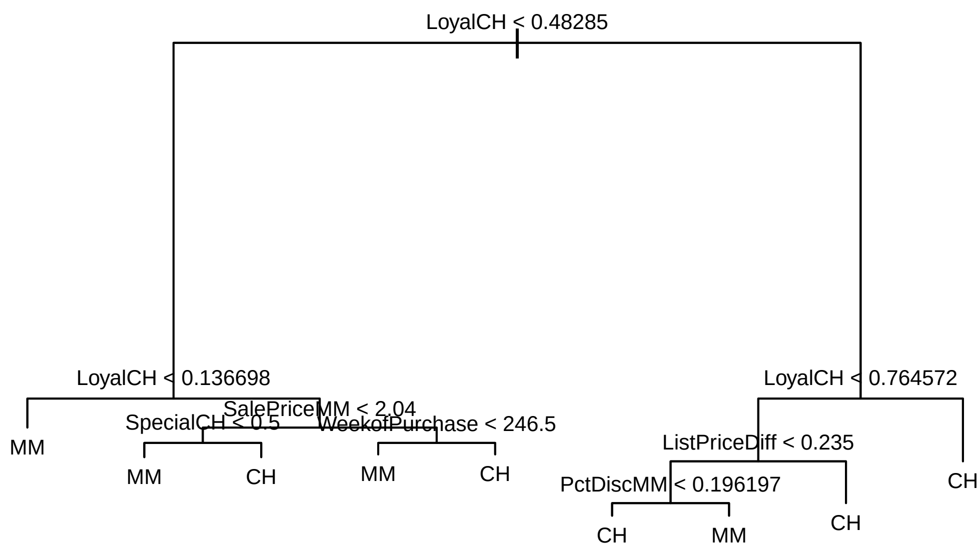
```r
tree_oj
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1072.00 CH ( 0.60750 0.39250 )
##    2) LoyalCH < 0.48285 305  318.60 MM ( 0.21639 0.78361 )
##      4) LoyalCH < 0.136698 103   45.76 MM ( 0.05825 0.94175 ) *
##      5) LoyalCH > 0.136698 202  245.80 MM ( 0.29703 0.70297 )
##       10) SalePriceMM < 2.04 115  112.30 MM ( 0.19130 0.80870 )
##         20) SpecialCH < 0.5 102   81.59 MM ( 0.13725 0.86275 ) *
##         21) SpecialCH > 0.5 13   17.32 CH ( 0.61538 0.38462 ) *
##       11) SalePriceMM > 2.04 87  119.20 MM ( 0.43678 0.56322 )
##         22) WeekofPurchase < 246.5 21   17.22 MM ( 0.14286 0.85714 ) *
##         23) WeekofPurchase > 246.5 66   91.25 CH ( 0.53030 0.46970 ) *
##    3) LoyalCH > 0.48285 495  421.10 CH ( 0.84848 0.15152 )
##      6) LoyalCH < 0.764572 236  277.80 CH ( 0.72458 0.27542 )
##       12) ListPriceDiff < 0.235 91  126.10 MM ( 0.49451 0.50549 )
##         24) PctDiscMM < 0.196197 72   97.80 CH ( 0.58333 0.41667 ) *
##         25) PctDiscMM > 0.196197 19   16.57 MM ( 0.15789 0.84211 ) *
##       13) ListPriceDiff > 0.235 145  112.60 CH ( 0.86897 0.13103 ) *
##      7) LoyalCH > 0.764572 259   84.69 CH ( 0.96139 0.03861 ) *
```

Branches that lead to terminal nodes are indicated by astrisk symbols. In each row, we can see the split criterion, and the number of observations, the deviance and the overall prediction for the branch. The first number in parentheses is the fraction of observations that take on the value `MM` and the second is the fraction that take on the value `CH`.

Part 9.d)

```r
plot(tree_oj)
text(tree_oj, pretty = FALSE, cex = 0.7)
```



According to the tree we grew, brand loyalty to CH is the most important predictor for sales. Given `LoyalCH` is greater than 0.75 or smaller than 0.027, other variables seem to play little role in determining `Purchase`. But among customers who do not have much loyalty to either brand, price difference does influence purchases, and if price of MM is low enough, compared to CH, they tend to buy MM. Loyalty matters even among these medium-loyalty customers: it takes a much cheaper MM (`PriceDiff` $<$ -0.165) to persuade someone a bit loyal to CH (0.5 $<$ `LoyalCH` $<$ 0.75) to buy MM (compared to `PriceDiff` $<$ 0.05 for 0.28 $<$ `LoyalCH` $<$ 0.5).

Part 9.e)

```
preds_oj = predict(tree_oj, newdata = OJ[test, ], type = "class")
table_oj <- table(test_purchase, preds_oj)
table_oj
```

```
##              preds_oj
## test_purchase  CH  MM
##            CH 155  12
##            MM  46  57
```

Test error rate is about 19%:

```
1 - sum(diag(table_oj))/sum(table_oj)
```

```
## [1] 0.2148148
```

```
mean(test_purchase != preds_oj)
```

```
## [1] 0.2148148
```

Part 9.f)

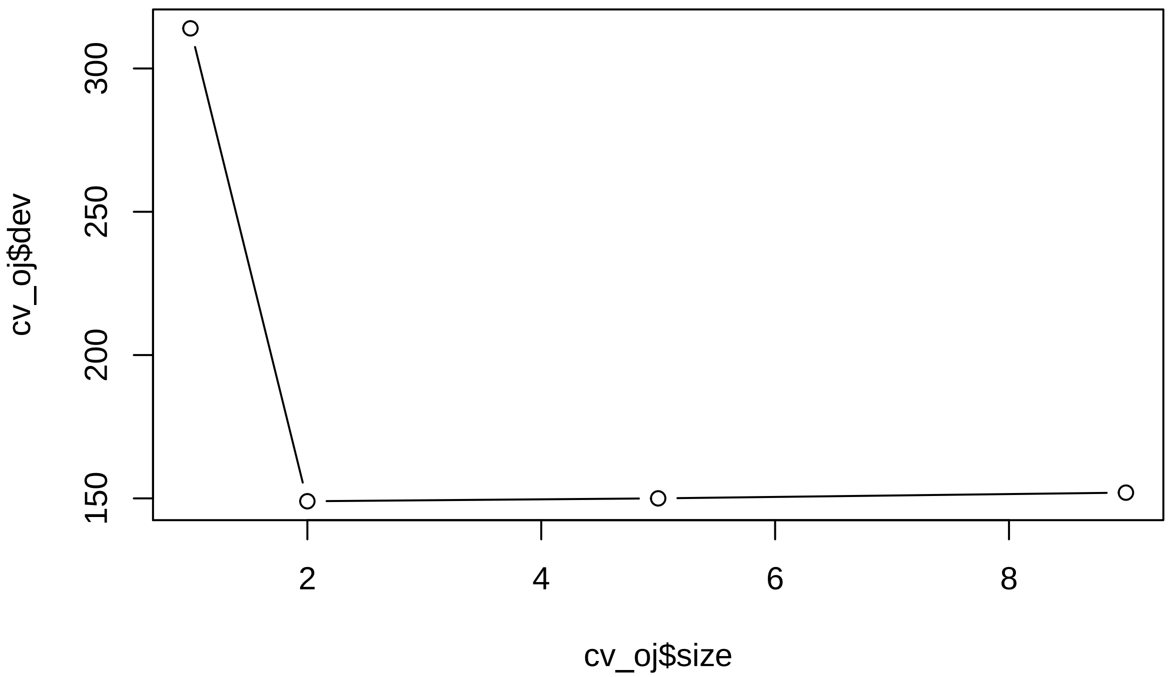
```r
set.seed(1)
(cv_oj = cv.tree(tree_oj, FUN = prune.misclass))
```

```
## $size
## [1] 9 5 2 1
##
## $dev
## [1] 152 150 149 314
##
## $k
## [1]       -Inf   1.750000   4.333333 173.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

k represents the tuning parameter in the minimization problem solved in cost-complexity pruning. When k is equal to ∞, it is as if we are maximizing $|T|$ which gives us the tree that we start with, the one with $T_0|$ terminal nodes. k is represented as α in ISLR.
In the example above, the sequence of trees T_α above, found by solving the cost-complexity minimization, are a subset of trees resulted by weakest link pruning. Although weakest link pruning yields all subsets of the original tree (i.e. sub-trees with sizes equal to 8, 7, 6, 5, 4, 3, 2, 1), only a subset of them (trees with size equal to 9, 5, 2, 1) solve the cost-complexity problem.

Part 9.g)

```r
plot(cv_oj$size, cv_oj$dev, type = "b")
```



Part 9.h)

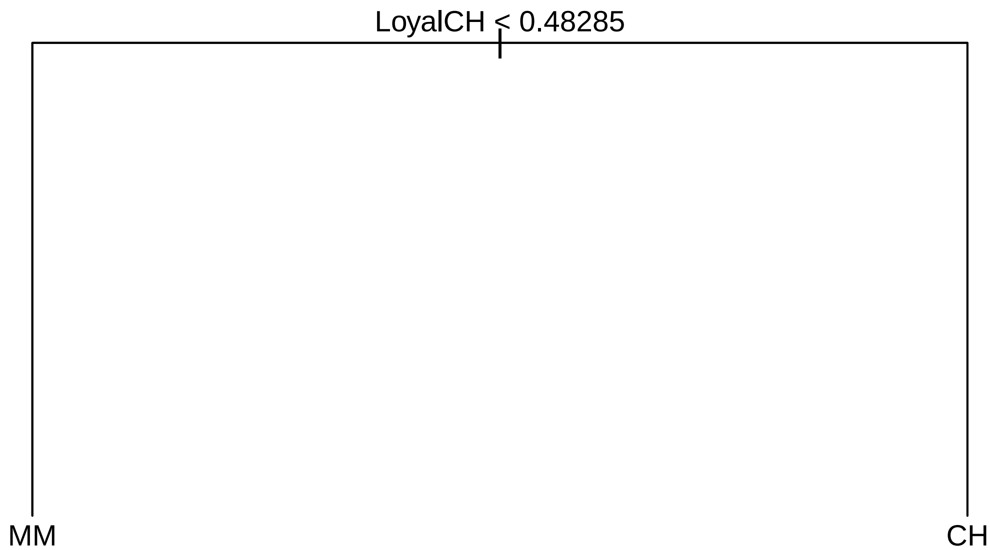
The error rate (confusingly labelled above as `dev` in the output) is minimized for a tree with 2 terminal nodes. But the difference in test error rates for trees of size 2, 5 and 8 is very small.

Part 9.i)

All the trees we found above by using cost-complexity pruning (done by `cv.tree`) are a subset of sub-trees that can be found using weakest link pruning (done by `prune.tree`). Hence, any tree with a given size $|T|$ is a tree found by weakest link pruning (successively ommiting branches) if and only if it is found by cost-complexity pruning (solving the optimization problem).

As a result, given any tree, all we need to know about the the optimal sub-tree is the size of it. Without knowing anything about the shape of the optimal sub-tree (beyond its size), we can easily find it through weakest link pruning.

```
prune_oj = prune.tree(tree_oj, best = 2)
plot(prune_oj)
text(prune_oj, pretty = FALSE, cex = 0.9)
```



Part 9.j)

The training and test error rates should be the same, since the predicted values for trees with size 6 and 8 are the same. The split that is omitted by pruning led to the same predicted value for both leaves (it only existed since it increased node purity).
We already showed that the classification test error rates are the same for the full tree and the pruned one. Below, we confirm the training error rates are also the same for them:

```r
# training error rate for full tree
preds_full <- predict(tree_oj, newdata = OJ[train, ], type = "class")
mean(OJ$Purchase[train] != preds_full)
```

```
## [1] 0.15125
```

```r
# training error rate for optimal tree
preds_prune <- predict(prune_oj, newdata = OJ[train, ], type = "class")
mean(OJ$Purchase[train] != preds_prune)
```

```
## [1] 0.17625
```

The full tree has a smaller test error rate, which is not strange, given very similar C.V. error rates between the full tree and the pruned one.

Part 9.k)

See part 9.j.

Exercise 10

Knowing the data:

```
library(ISLR)
# know the data
dim(Hitters)
```

```
## [1] 322  20
```

```
summary(Hitters)  # League, Division and NewLeague are quantitative
```

```
##      AtBat            Hits          HmRun            Runs
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00
##
##       RBI             Walks            Years            CAtBat
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
##
##      CHits            CHmRun           CRuns             CRBI
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00
##
##      CWalks          League  Division    PutOuts          Assists
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0
##  Median : 170.50                    Median : 212.0   Median : 39.5
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0
##
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146
##  Median : 6.00   Median : 425.0
##  Mean   : 8.04   Mean   : 535.9
##  3rd Qu.:11.00   3rd Qu.: 750.0
##  Max.   :32.00   Max.   :2460.0
##                  NA's   :59
```

```
sum(is.na(Hitters))  # 59 missing
```

```
## [1] 59
```

```r
# know the dependent variable: annual salary in thousands of dollars (see ?Hitters)
sum(is.na(Hitters$Salary))  # 59 missing
```

```
## [1] 59
```

```r
hist(Hitters$Salary)  # very skewed, long tail
```

Here, we know the missing values arise from `Hitters$Salary`. Nonetheless, in general for seeing what variables have missing values, we can also run the code below:

```r
length(Hitters)
```

```
## [1] 20
```

```r
sapply(Hitters, FUN = function(x) sum(is.na(x)))
```

```
##     AtBat      Hits     HmRun      Runs       RBI     Walks     Years
##         0         0         0         0         0         0         0
##    CAtBat     CHits    CHmRun     CRuns      CRBI    CWalks    League
##         0         0         0         0         0         0         0
##  Division   PutOuts   Assists    Errors    Salary NewLeague
##         0         0         0         0        59         0
```

Part 10.a)

Cleaning the data:

```r
# we omit the missing, as suggested in the exercise:
Hits <- na.omit(Hitters)
# log-transform salary:
Hits$lsalary <- log(Hits$Salary)
hist(Hits$lsalary)
```

The log-transformation alleviates the skewness to a great extent. From now on, we work with `Hits` as the data and `lsalary` as the response.

Part 10.b)

```r
train = 1:200
test = -train
```

Part 10.c)

We want to draw training MSE as a function of the shrinkage parameter.

```r
# grid for lambda
(lambda_grid <- 10^seq(0, -4, length = 9))
```

```
## [1] 1.0000000000 0.3162277660 0.1000000000 0.0316227766 0.0100000000
## [6] 0.0031622777 0.0010000000 0.0003162278 0.0001000000
```

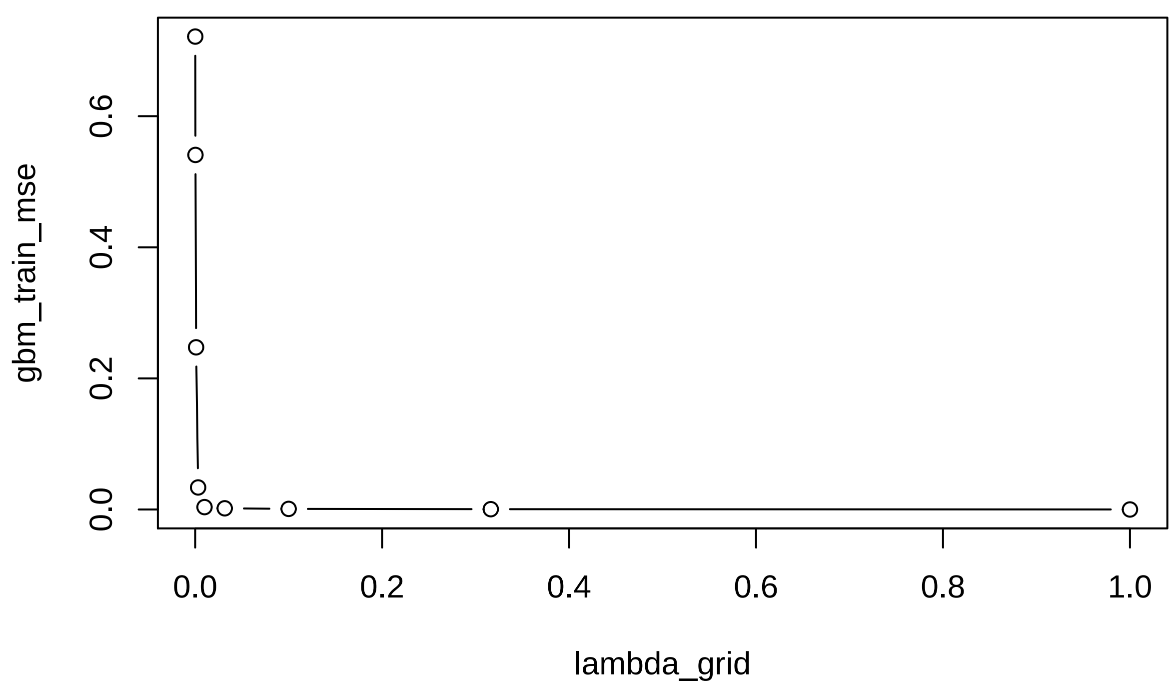
```r
# training MSE
library(gbm)
```

```
## Loading required package: survival

## Loading required package: lattice

## Loaded gbm 2.1.3
```

```r
library(ggplot2)
set.seed(1)
gbm_fits <- vector(mode = "list", length = length(lambda_grid))
gbm_train_preds <- vector(mode = "list", length = length(lambda_grid))
gbm_train_mse <- vector(length = length(lambda_grid))
for (i in 1:length(lambda_grid)) {
  gbm_fits[[i]] <- gbm(lsalary ~ ., data = Hits[train, ], distribution = "gaussian",
         n.trees = 1000, interaction.depth = 1, shrinkage = lambda_grid[i])
  gbm_train_preds[[i]] <- predict(gbm_fits[[i]], newdata = Hits[train, ], n.trees = 1000)
  gbm_train_mse[i] <- mean((Hits$lsalary[train] - gbm_train_preds[[i]])^2)
}
```

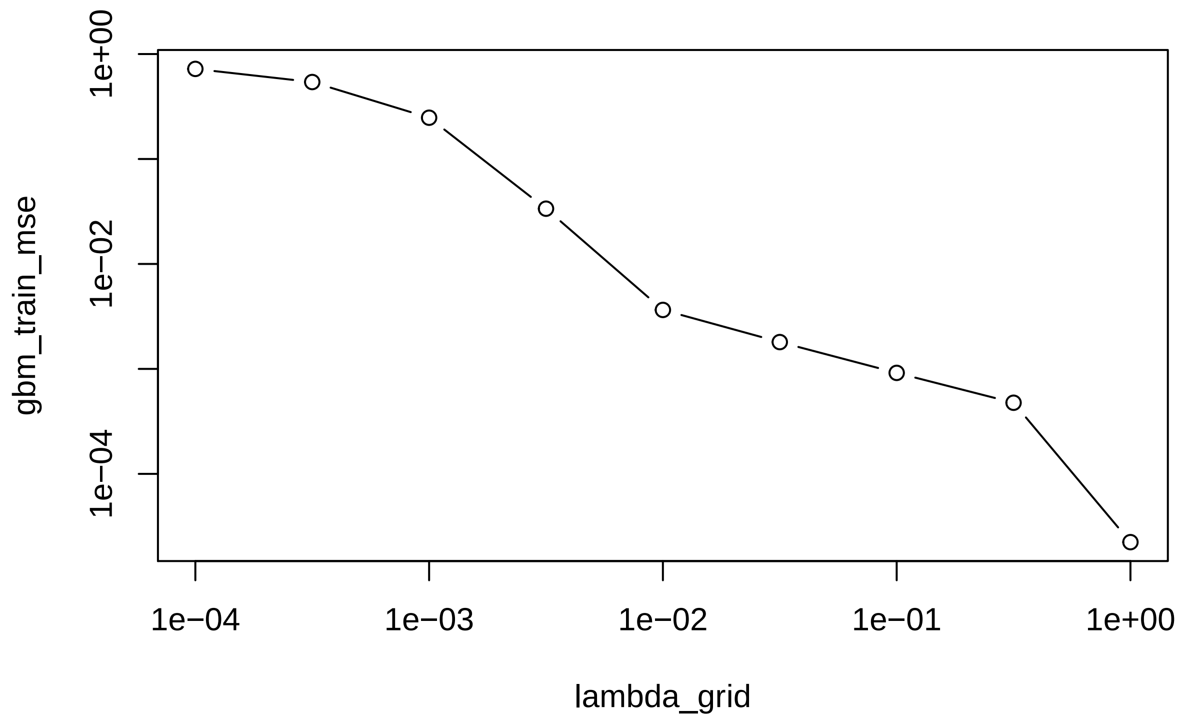
Here is the plot:

```r
plot(lambda_grid, gbm_train_mse, type = "b")
```



As we see above the training MSE goes down to zero very quickly as we decrease λ. The direction of change in the training MSE could be seen more clearly by using a log-log scale:

```r
plot(lambda_grid, gbm_train_mse, log = "xy", type = "b")
```



The log-log transformation shows the percentage change in training MSE as we change λ. Training MSE goes down as we increase λ. What the above plots show is that we get a better fit to the training data as we decrease the rate of learning. But the result is probably due to fixing the number of trees; for larger values of λ we would need fewer trees to avoid overfitting. But since we are fiing the number of trees, the decline in the training MSE is likely to be due to overfitting for large enough shrinkage parameters. We need the compute the test MSE for finding the optimal value of λ.

As we saw, in this example it was easier to use the base plotting function in R for our purpose. Nevertheless, `ggplot` enables the implementation of much more elaborate details. For example, see here. –>

Part 10.d)

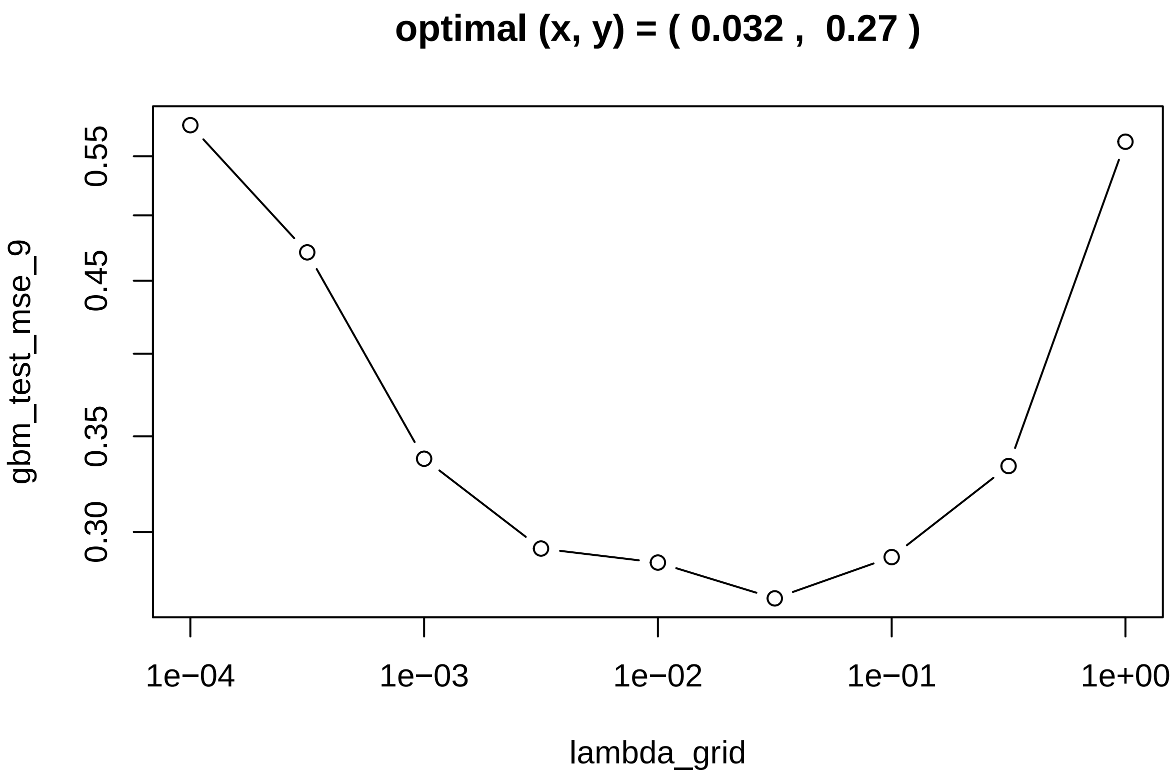
To enable working with different combinations, we write a function:

```r
test_mse_ex10 <-
  function(data, lambda.grid, n.trees) {
    set.seed(1)
    gbm_test_mse <- rep(NA, length = length(lambda.grid))
      for (i in 1:length(lambda.grid)) {
        gbm_fit <- gbm(lsalary ~ . - Salary - lsalary, data = data[train, ], distribution = "gaussian",
                  shrinkage = lambda.grid[i], interaction.depth = 1, n.trees = n.trees)
        gbm_preds <- predict(gbm_fit, newdata = data[test, ], n.tree = n.trees)
        gbm_test_mse[i] <- mean((data$lsalary[test] - gbm_preds)^2)
      }
```

```
  gbm_test_mse
  }
```

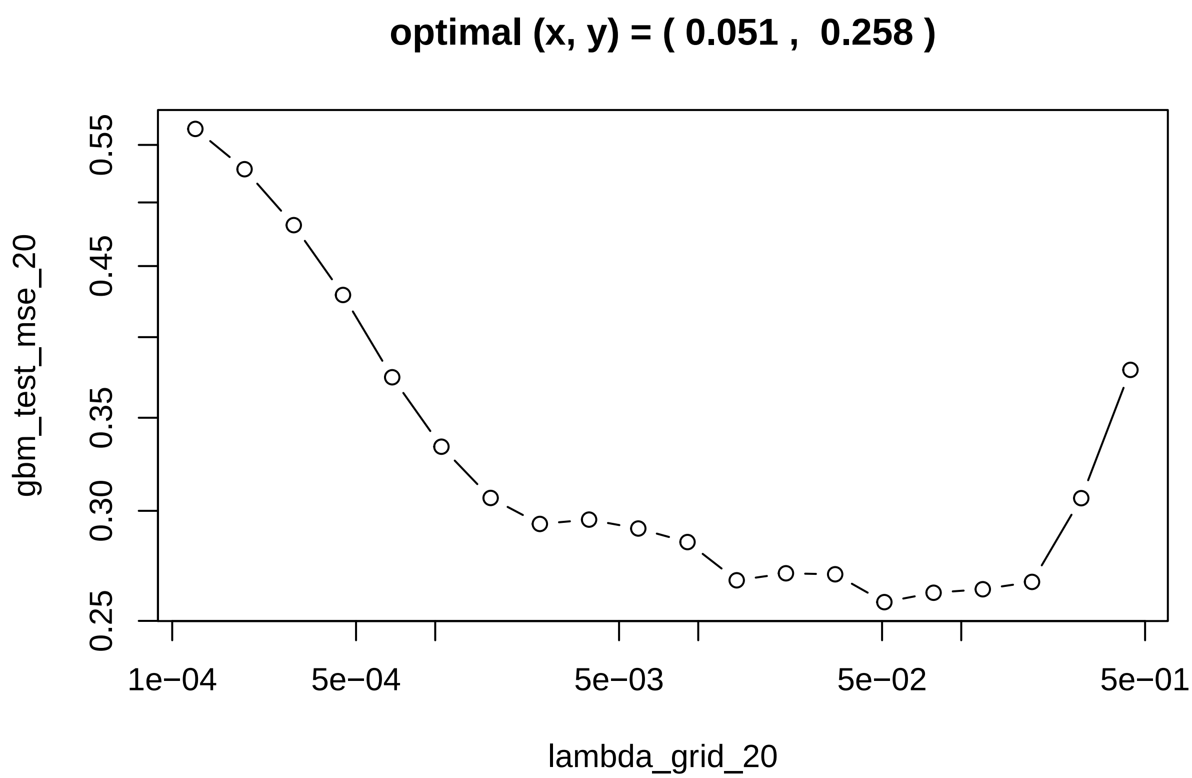
We use the same grid for the test MSE.

```
gbm_test_mse_9 <- test_mse_ex10(Hits, lambda.grid = lambda_grid, n.trees = 1000)
opt_mse_9 <- c(lambda_grid[gbm_test_mse_9 == min(gbm_test_mse_9)],
             min(gbm_test_mse_9))
plot(lambda_grid, gbm_test_mse_9, log = "xy", type = "b")
title(paste("optimal (x, y) = (", round(opt_mse_9[1], 3), ", ", round(opt_mse_9[2], 3), ")"))
```



We can prove that the minimum in the log-log plot is the same as the minimum in the non-transformed plot, since x and y are positive (it can be shown using the fact that $\frac{d\log(y)}{d\log(x)} = \frac{dy}{dx} \times \frac{x}{y}$, so whenever the derivative of the non-transformed function changes sign from negative to positive, so the does the derivative of transfromed). Since we do not have many grid points, the optimal point might not be the minimum point in the plot above. Below, we use a finer grid, but noly only increasing the number of grid points to 20, but also narrowing our focus on a smaller region:

```
lambda_grid_20 <- 10^seq(from = log(0.02), to = log(0.7), length = 20)
gbm_test_mse_20 <- test_mse_ex10(Hits, lambda.grid = lambda_grid_20, n.trees = 1000)
opt_mse_20 <- c(lambda_grid_20[gbm_test_mse_20 == min(gbm_test_mse_20)],
             min(gbm_test_mse_20))
plot(lambda_grid_20, gbm_test_mse_20, log = "xy", type = "b")
title(paste("optimal (x, y) = (", round(opt_mse_20[1], 3), ", ", round(opt_mse_20[2], 3), ")"))
```




```
c(lambda_grid_20[gbm_test_mse_20 == min(gbm_test_mse_20)], min(gbm_test_mse_20))
```

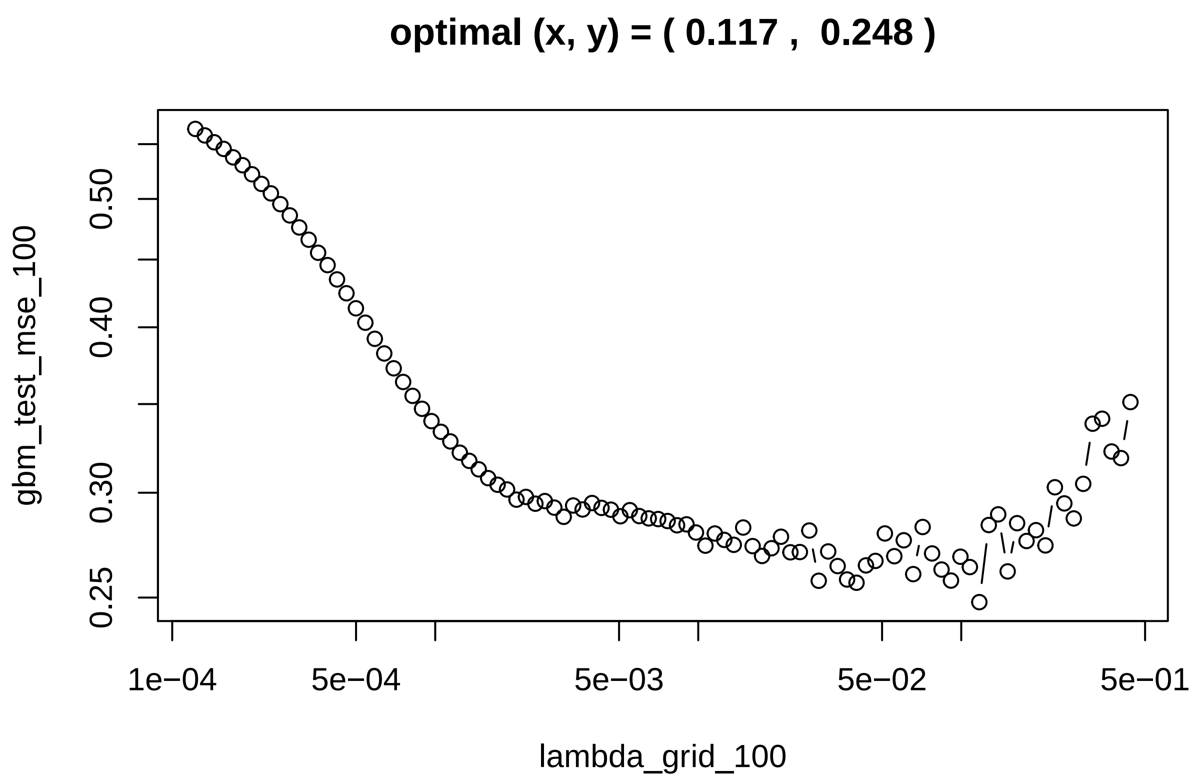
```
## [1] 0.05101609 0.25788970
```

An interesting point is that the test MSE seems to be sensitive to the amount of λ, in the sense that a small change in λ may casue a relatively large change in the test MSE. This is better seen using a finer grid, which we will investigate in the next subsection. However, we will use the amount of λ we find here in the next rest of the exercise.

Digression: the effect of finder grids

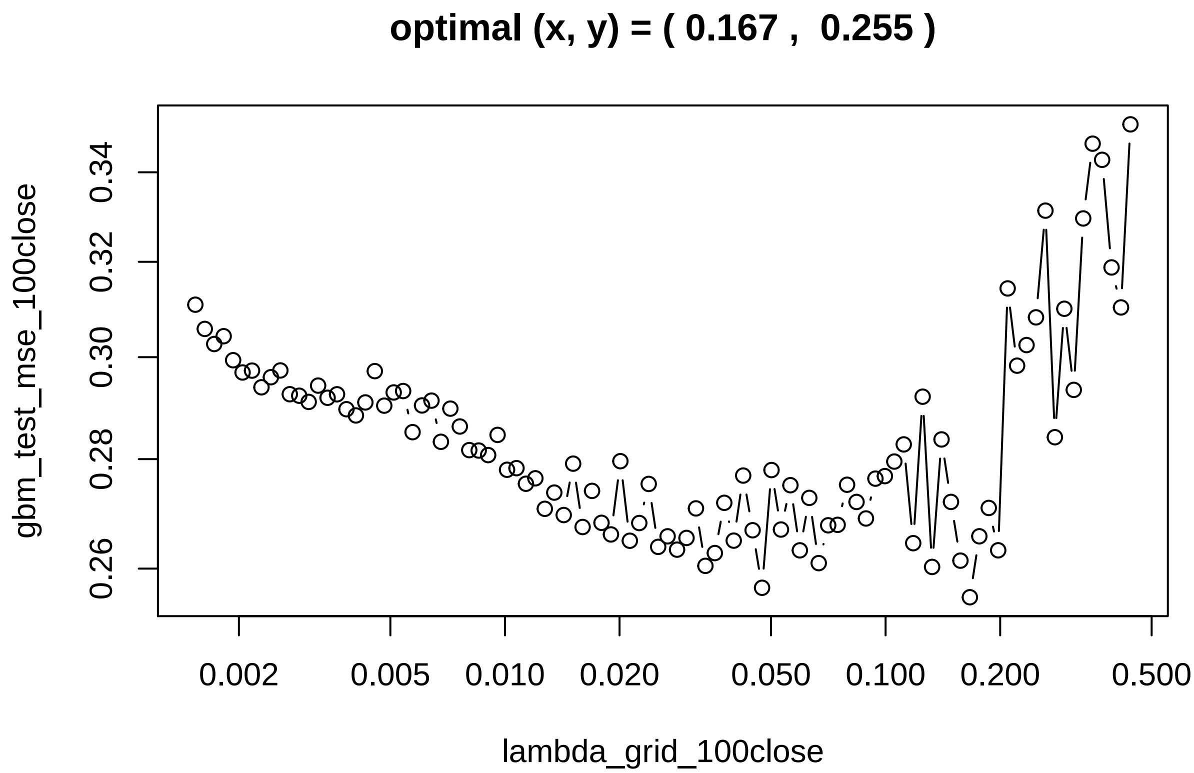
The picture below shows the change in the plot if we only increase the number of grid points from 20 to 100, given the same interval:

```
lambda_grid_100 <- 10^seq(from = log(0.02), to = log(0.7), length = 100)
gbm_test_mse_100 <- test_mse_ex10(Hits, lambda.grid = lambda_grid_100, n.trees = 1000)
opt_mse <- c(lambda_grid_100[gbm_test_mse_100 == min(gbm_test_mse_100)],
             min(gbm_test_mse_100))
plot(lambda_grid_100, gbm_test_mse_100, log = "xy", type = "b")
title(paste("optimal (x, y) = (", round(opt_mse[1], 3), ", ", round(opt_mse[2], 3), ")"))
```



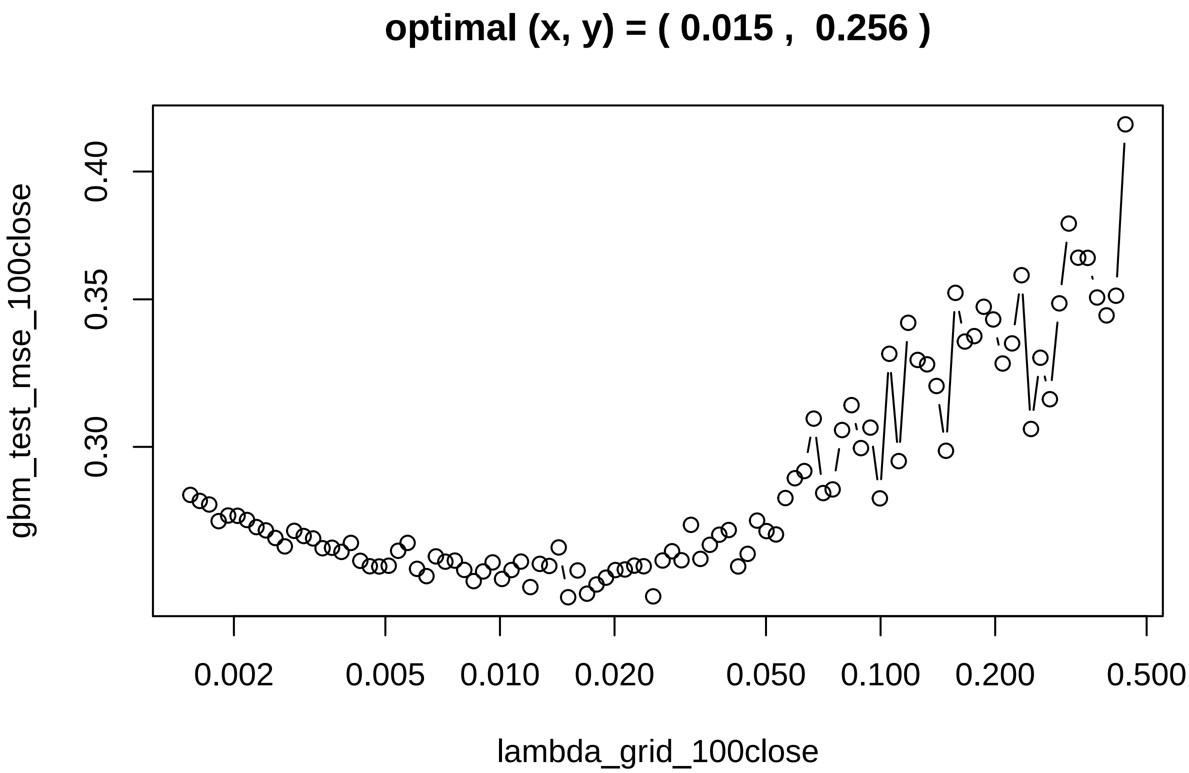
Below we look even closer. Even a closer look shows us that the optimal λ should be between 0.02 and 0.2. But, due to the fluctuations, it does not give us much information beyond that. So we keep the number of grid points equal to 100, but narrow the interval:

```
lambda_grid_100close <- 10^seq(from = log(0.06), to = log(0.7), length = 100)
gbm_test_mse_100close <-
  test_mse_ex10(Hits, lambda.grid = lambda_grid_100close, n.trees = 1000)
opt_mse <- c(lambda_grid_100close[gbm_test_mse_100close == min(gbm_test_mse_100close)],
             min(gbm_test_mse_100close))
plot(lambda_grid_100close, gbm_test_mse_100close, log = "xy", type = "b")
title(paste("optimal (x, y) = (", round(opt_mse[1], 3), ", ", round(opt_mse[2], 3), ")"))
```



How would the optimal λ change when we increase the number of trees? Does it improve the test MSE? We keep the narrowest interval we experimated with, as the previous plot, and increase the number of trees from 1000 to 5000:

```
lambda_grid_100close <- 10^seq(from = log(0.06), to = log(0.7), length = 100)
gbm_test_mse_100close <- test_mse_ex10(Hits, lambda.grid = lambda_grid_100close, n.trees = 5000)
opt_mse <- c(lambda_grid_100close[gbm_test_mse_100close == min(gbm_test_mse_100close)], min(gbm_test_mse
plot(lambda_grid_100close, gbm_test_mse_100close, log = "xy", type = "b")
title(paste("optimal (x, y) = (", round(opt_mse[1], 3), ", ", round(opt_mse[2], 3), ")"))
```



In this example, optimal λ decreases when we increase the number of trees, which is in line with the idea that when we have more trees, we can have slower pace of learning.

The test MSE is very close in all cases we studied above, except for the case where we had only 9 grid points. Hence, in this example, we would have been fine as long as we used a moderate number of grid points. However, note that this is a very small dataset and we are using only 63 observations for the test data.

Part 10.e)

Chapter 3 covered linear regression and chapter 6 covered subset selection, shrinkage and dimension reduction methods. The questions asks for combining linear regression with either of the methods in chapter 6. We will work only with linear variables, so will not make any higher-order transformation.

Best subset selection

We use cross-validation with 5 folds.

```
best_size1 <- which.min(Cv_glm)
```

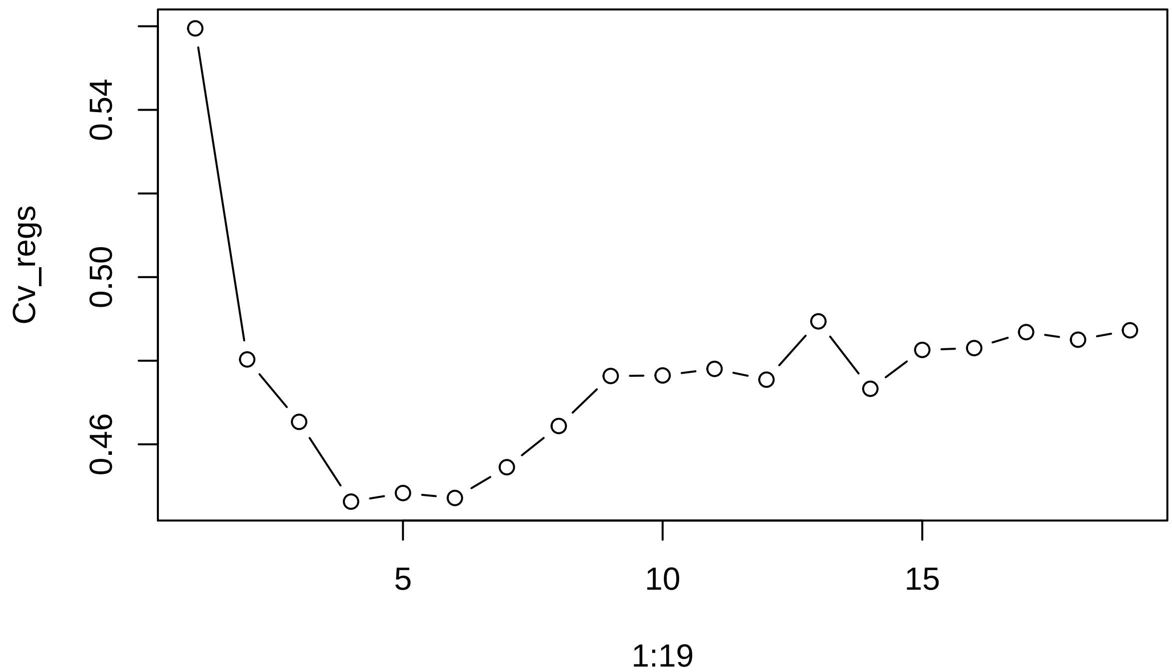
The warnings arise from `cv.glm()` and the fact that it find the number of obervations too small. This is a problem we cannot address unless we have more data.

To understand why this is wrong, suppose we were using validation set approach instead of cross-validation. This is like using both training and test data for finding the best size and then refitting the model on the training data and evaluating on the test data. The problem is that we are using the test data for the model selection, which introduces overfitting. Hece, when using cross-validation, the test data in each fold could not be used in any way for training.

Second method

–>

```
library(leaps)
set.seed(1)
## define the predict function for regsubsets:
predict.regsubsets =
  function (object, newdata ,id, ...) {
    form = as.formula(object$call[[2]])
    mat = model.matrix(form, newdata)
    coefi = coef(object, id = id)
    xvars = names(coefi)
    mat[,xvars]%*%coefi
  }
## Compute CV error
# define folds
folds <- sample(1:5, size = length(train), replace = TRUE)
# to avoid confusion of training set with different training data in cv
Hits_cv <- Hits[train, ]
Cv_mat <- matrix(NA, nrow = 19, ncol = 5)
for (j in 1:5) {
  regfit_j <- regsubsets(lsalary ~ . - Salary, data = Hits_cv[folds != j, ], nvmax = 19)
  for (i in 1:19) {
    preds_ij <- predict(regfit_j, Hits_cv[folds == j, ], id = i)
    Cv_mat[i, j] <- mean((preds_ij - Hits_cv$lsalary[folds == j])^2)
  }
}
Cv_regs <- apply(Cv_mat, FUN = mean, MARGIN = 1)
plot(1:19, Cv_regs, type = "b")
```



The plot above implies that the optimal size is 4, i.e. with four variables. Now, we should retrieve the best model of size 4 using full training data (so far we have only used 4 folds out of 5 as training data in cross-validation):

```
regfit_full <- regsubsets(lsalary ~ . - Salary, data = Hits[train, ],
                          nvmax = 4)  # four variables is enough
preds2 <- predict(regfit_full, newdata = Hits[test, ], id = 4)
(mse_best2 <- mean((preds2 - Hits$lsalary[test])^2))
```

```
## [1] 0.4942052
```

Ridge regression

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## foreach: simple, scalable parallel programming from Revolution Analytics
## Use Revolution R for scalability, fault tolerance and more.
## http://www.revolutionanalytics.com
```

```
## Loaded glmnet 2.0-10
```

```
Hits_x = model.matrix(lsalary ~ . - Salary, data = Hits)[, -1]
lambda_grid <- 10^seq(10, -2, length = 100)
fit_ridge <- glmnet(x = Hits_x[train, ], y =  Hits$lsalary[train], alpha = 0,
```

```
       lambda = lambda_grid, thresh = 1e-12)
cv_ridge <- cv.glmnet(x = Hits_x[test, ], y = Hits$lsalary[test], alpha = 0)
best_lam <- cv_ridge$lambda.min
preds_ridge <- predict(fit_ridge, s = best_lam, newx = Hits_x[test, ])
(mse_ridge <- mean((preds_ridge - Hits$lsalary[test])^2))
```

```
## [1] 0.4424325
```

Lasso regression

```
library(glmnet)
Hits_x = model.matrix(lsalary ~ . - Salary, data = Hits)[, -1]
lambda_grid <- 10^seq(10, -2, length = 100)
fit_lasso <- glmnet(x = Hits_x[train, ], y =  Hits$lsalary[train], alpha = 1,
       lambda = lambda_grid, thresh = 1e-12)
cv_lasso <- cv.glmnet(x = Hits_x[test, ], y = Hits$lsalary[test], alpha = 1)
best_lam <- cv_lasso$lambda.min
preds_lasso <- predict(fit_lasso, s = best_lam, newx = Hits_x[test, ])
(mse_lasso <- mean((preds_lasso - Hits$lsalary[test])^2))
```

```
## [1] 0.4398225
```

PCR

```
library(pls)
```

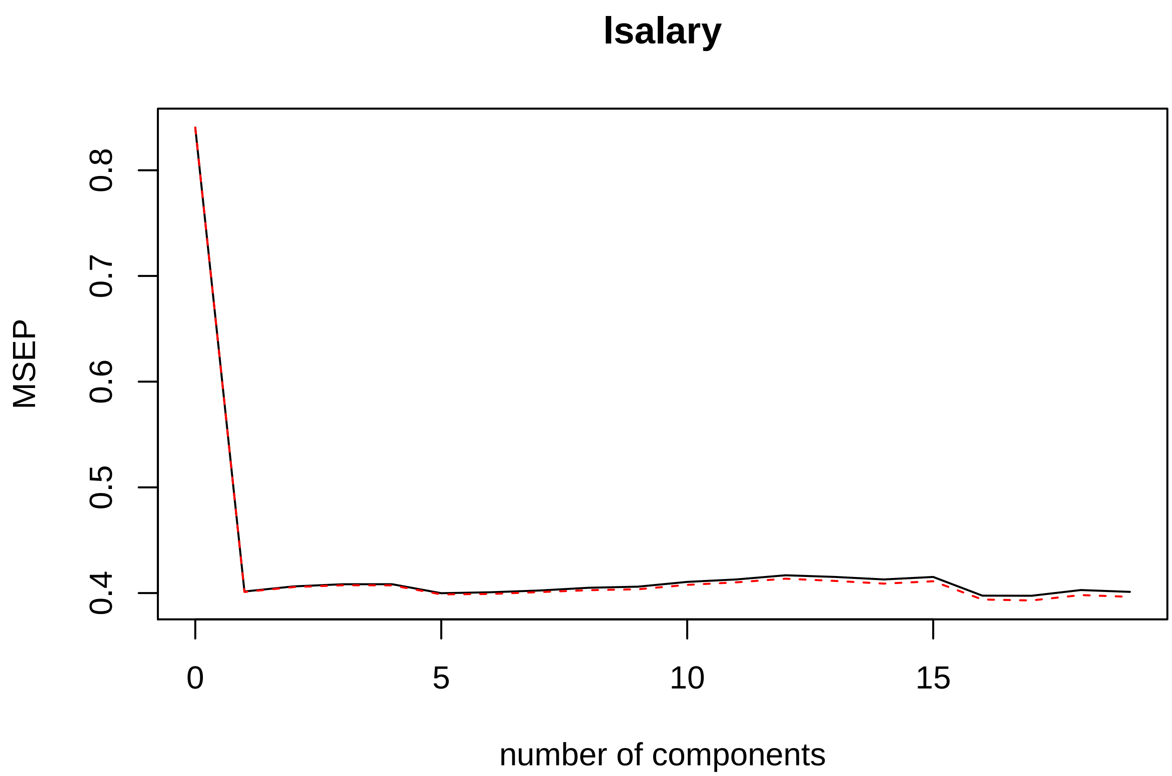
```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```
pcr_fit <- pcr(lsalary ~ . - Salary, data = Hits[train, ],
    scale = TRUE, validation = "CV")
summary(pcr_fit)
```

```
## Data:    X dimension: 200 19
##  Y dimension: 200 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9167   0.6336   0.6374   0.6390   0.6390   0.6324   0.6330
## adjCV       0.9167   0.6333   0.6369   0.6383   0.6382   0.6315   0.6318
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.6344   0.6364   0.6372    0.6407    0.6426    0.6456    0.6444
## adjCV   0.6332   0.6346   0.6353    0.6385    0.6404    0.6431    0.6415
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV       0.6426    0.6444    0.6305    0.6304    0.6347    0.6333
## adjCV    0.6395    0.6412    0.6276    0.6269    0.6310    0.6296
```

```
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X          39.40    60.67    70.76    79.64    84.73    89.29    92.38
## lsalary    52.42    52.67    53.22    53.56    54.56    55.34    55.70
##          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X          95.02    96.32     97.26     98.06     98.72     99.23
## lsalary    56.48    56.68     56.77     56.80     57.41     57.91
##          14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X           99.54     99.77     99.90     99.97     99.99    100.00
## lsalary     58.68     58.75     60.27     61.27     61.27     61.49
```

```
validationplot(pcr_fit, val.type = "MSEP")
```



Choosing 1 component gives a reasonable cross-validation error.

```
preds_pcr <- predict(pcr_fit, newdata = Hits[test, ], ncomp = 1)
(cv_pcr <- mean((preds_pcr - Hits$lsalary[test])^2))
```

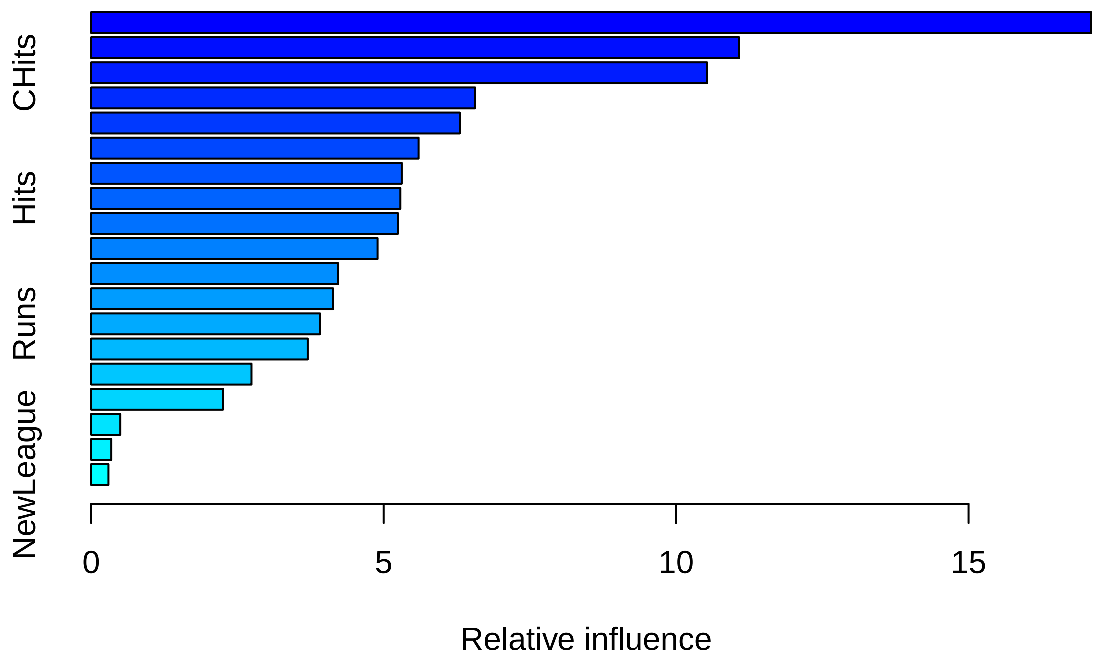
```
## [1] 0.4661183
```

The performance of boosting is impressive: it leads to test MSE estimate of about 0.26, while the best performance we get from the combination of a linear model with methods of chapter 6 is more than 0.43. This is despite the fact that we use an additive boosing model (we use stumps). This is an indication of nonlinearity in the data.

Part 10.f)

We use the value of λ we found with 20 grids, and use the full data:

```
best_lam <- lambda_grid_20[gbm_test_mse_20 == min(gbm_test_mse_20)]
gbm_fit <- gbm(lsalary ~ . - Salary - lsalary, data = Hits, distribution = "gaussian",
        shrinkage = best_lam, interaction.depth = 1, n.trees = 1000)
summary(gbm_fit)
```




```
##                   var    rel.inf
## CAtBat         CAtBat 17.0952907
## CRBI             CRBI 11.0769797
## CHits           CHits 10.5285551
## CWalks         CWalks  6.5644198
## CRuns           CRuns  6.3014537
## Years           Years  5.5972824
## Walks           Walks  5.3090311
## Hits             Hits  5.2856063
## CHmRun         CHmRun  5.2416846
## PutOuts       PutOuts  4.8958056
## RBI               RBI  4.2241857
## HmRun           HmRun  4.1355981
## Runs             Runs  3.9128950
## AtBat           AtBat  3.7021136
## Errors         Errors  2.7403018
## Assists       Assists  2.2520485
## League         League  0.4980133
```

```
## Division   Division  0.3436503
## NewLeague NewLeague  0.2950848
```

`CAtBat` appears to be the most important variable. Perhaps more interestingly, the variables that measure values during the career seem to be much more important that the same-season measurements.

Part 10.g)

The bagging test MSE:

```r
library(randomForest)
rf_fit <- randomForest(lsalary ~ . - Salary, data = Hits[train, ],
                       mtry = 19, ntree = 500, importance = TRUE)
rf_preds <- predict(rf_fit, newdata = Hits[test, ])
(rf_mse <- mean((rf_preds - Hits$lsalary[test])^2))
```

```
## [1] 0.22887
```

The test MSE is slightly smaller than the one we found for boosting. This might be because of using an additive model for boosting, while the bagging estimator is more flexible.

Exercise 11

```r
library(ISLR)
# the data
dim(Caravan)  # 86 variables
```

```
## [1] 5822   86
```

```r
sum(is.na(Caravan))
```

```
## [1] 0
```

```r
# response
summary(Caravan$Purchase)
```

```
##   No  Yes
## 5474  348
```

```r
contrasts(Caravan$Purchase)
```

```
##     Yes
## No    0
## Yes   1
```

`gbm()` with a qualitative variable as response requires transformation to a 0-1 dummy variable:

```r
Caravan$Purchase <- ifelse(Caravan$Purchase == "Yes", 1, 0)
summary(Caravan$Purchase)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05977 0.00000 1.00000
```

Part 11.a)

```r
train = 1:1000
test = -train
```

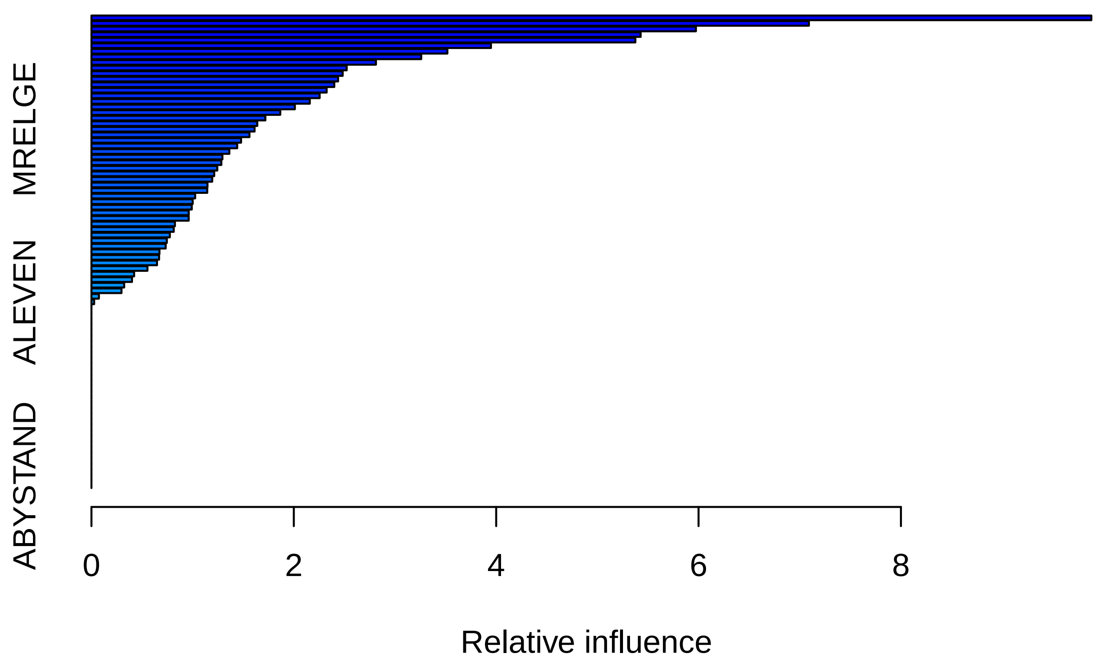
The test set is nearly 4 times larger than the training set.

Part 11.b)

```r
library(gbm)
set.seed(1)
gbm_fit <- gbm(Purchase ~ ., data = Caravan[train, ], distribution = "bernoulli",
               interaction.depth = 2, n.trees = 1000, shrinkage = 0.01)
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 50: PVRAAUT has no variation.

## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 71: AVRAAUT has no variation.
```

```r
summary(gbm_fit)
```




```
##               var    rel.inf
## PPERSAUT PPERSAUT 9.88196755
## MKOOPKLA MKOOPKLA 7.09057539
## MOPLHOOG MOPLHOOG 5.97383721
```

```
## PBRAND     PBRAND 5.42795341
## MGODGE     MGODGE 5.37533300
## MBERMIDD MBERMIDD 3.94847387
## MINK3045 MINK3045 3.51763123
## MGODPR     MGODPR 3.25916272
## MOSTYPE   MOSTYPE 2.81168103
## MBERARBG MBERARBG 2.52290762
## ABRAND     ABRAND 2.48318472
## MAUT2       MAUT2 2.43784324
## MSKA         MSKA 2.39996975
## MAUT1       MAUT1 2.32463277
## MSKC         MSKC 2.25575413
## PWAPART   PWAPART 2.15817221
## MSKB1       MSKB1 2.01054800
## MINK7512 MINK7512 1.86610130
## MFGEKIND MFGEKIND 1.71887449
## MFWEKIND MFWEKIND 1.63896537
## MRELGE     MRELGE 1.61226288
## MBERHOOG MBERHOOG 1.56162484
## MGODOV     MGODOV 1.47899843
## MINKGEM   MINKGEM 1.44042382
## MZFONDS   MZFONDS 1.36297740
## MOPLMIDD MOPLMIDD 1.29452540
## MGODRK     MGODRK 1.28364889
## APERSAUT APERSAUT 1.24368912
## MINK4575 MINK4575 1.21344684
## MRELOV     MRELOV 1.19368544
## MAUT0       MAUT0 1.14689401
## MHHUUR     MHHUUR 1.14420398
## MGEMLEEF MGEMLEEF 1.02514716
## MZPART     MZPART 1.00085325
## MHKOOP     MHKOOP 0.99160310
## MINKM30   MINKM30 0.96171242
## MOSHOOFD MOSHOOFD 0.96138547
## PBYSTAND PBYSTAND 0.82501502
## PMOTSCO   PMOTSCO 0.81362486
## MFALLEEN MFALLEEN 0.77469168
## MBERARBO MBERARBO 0.74527101
## PLEVEN     PLEVEN 0.73389614
## MSKB2       MSKB2 0.67226643
## MGEMOMV   MGEMOMV 0.66848679
## MSKD         MSKD 0.64817413
## MRELSA     MRELSA 0.55338948
## MOPLLAAG MOPLLAAG 0.42198674
## MBERBOER MBERBOER 0.40081995
## MBERZELF MBERZELF 0.32375264
## MINK123M MINK123M 0.29723936
## MAANTHUI MAANTHUI 0.07381764
## ALEVEN     ALEVEN 0.02691667
## PWABEDR   PWABEDR 0.00000000
## PWALAND   PWALAND 0.00000000
## PBESAUT   PBESAUT 0.00000000
## PVRAAUT   PVRAAUT 0.00000000
## PAANHANG PAANHANG 0.00000000
```

```
## PTRACTOR PTRACTOR 0.00000000
## PWERKT     PWERKT 0.00000000
## PBROM       PBROM 0.00000000
## PPERSONG PPERSONG 0.00000000
## PGEZONG   PGEZONG 0.00000000
## PWAOREG   PWAOREG 0.00000000
## PZEILPL   PZEILPL 0.00000000
## PPLEZIER PPLEZIER 0.00000000
## PFIETS     PFIETS 0.00000000
## PINBOED   PINBOED 0.00000000
## AWAPART   AWAPART 0.00000000
## AWABEDR   AWABEDR 0.00000000
## AWALAND   AWALAND 0.00000000
## ABESAUT   ABESAUT 0.00000000
## AMOTSCO   AMOTSCO 0.00000000
## AVRAAUT   AVRAAUT 0.00000000
## AAANHANG AAANHANG 0.00000000
## ATRACTOR ATRACTOR 0.00000000
## AWERKT     AWERKT 0.00000000
## ABROM       ABROM 0.00000000
## APERSONG APERSONG 0.00000000
## AGEZONG   AGEZONG 0.00000000
## AWAOREG   AWAOREG 0.00000000
## AZEILPL   AZEILPL 0.00000000
## APLEZIER APLEZIER 0.00000000
## AFIETS     AFIETS 0.00000000
## AINBOED   AINBOED 0.00000000
## ABYSTAND ABYSTAND 0.00000000
```

`PPERSAUT` seems to be the most important determinant of `Purchase`.

Part 11.c)

```r
library(gmodels)
gbm_probs <- predict(gbm_fit, newdata = Caravan[test, ], n.trees = 1000, type = "response")
gbm_preds <- ifelse(gbm_probs > 0.2, 1, 0)
CrossTable(gbm_preds, Caravan$Purchase[test])
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  4822
##
##
```

```
##              | Caravan$Purchase[test]
##    gbm_preds |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      4359 |       252 |      4611 |
##              |     0.137 |     2.146 |           |
##              |     0.945 |     0.055 |     0.956 |
##              |     0.962 |     0.872 |           |
##              |     0.904 |     0.052 |           |
## -------------|-----------|-----------|-----------|
##            1 |       174 |        37 |       211 |
##              |     2.990 |    46.902 |           |
##              |     0.825 |     0.175 |     0.044 |
##              |     0.038 |     0.128 |           |
##              |     0.036 |     0.008 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      4533 |       289 |      4822 |
##              |     0.940 |     0.060 |           |
## -------------|-----------|-----------|-----------|
##
##
```

17.5% of the people predicted to make a purchase do in fact make one. It is more accurate than the KNN and logistic predictions below.

KNN

We try KNN with k = 2.

```r
library(class)
set.seed(1)
knn_preds <- knn(test = Caravan[test, ], train = Caravan[train, ], cl = Caravan$Purchase[train], k = 2)
CrossTable(knn_preds, Caravan$Purchase[test])
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  4822
##
##
##              | Caravan$Purchase[test]
##    knn_preds |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      4272 |       258 |      4530 |
##              |     0.043 |     0.671 |           |
##              |     0.943 |     0.057 |     0.939 |
```

```
##              |     0.942 |     0.893 |           |
##              |     0.886 |     0.054 |           |
## -------------|-----------|-----------|-----------|
##            1 |       261 |        31 |       292 |
##              |     0.664 |    10.413 |           |
##              |     0.894 |     0.106 |     0.061 |
##              |     0.058 |     0.107 |           |
##              |     0.054 |     0.006 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      4533 |       289 |      4822 |
##              |     0.940 |     0.060 |           |
## -------------|-----------|-----------|-----------|
##
##
```

10.6% of those who are predicted to purchase in fact purchase according to the KNN.

Logistic regression

We use the same threshold of 0.2 for logistic regression:

```
glm_fit <- glm(Purchase ~ ., data = Caravan[train, ], family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm_probs <- predict(glm_fit, newdata= Caravan[test, ], type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
glm_preds <- ifelse(glm_probs > 0.2, 1, 0)
CrossTable(glm_preds, Caravan$Purchase[test])
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  4822
##
##
##              | Caravan$Purchase[test]
##    glm_preds |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      4183 |       231 |      4414 |
##              |     0.271 |     4.254 |           |
##              |     0.948 |     0.052 |     0.915 |
##              |     0.923 |     0.799 |           |
##              |     0.867 |     0.048 |           |
```

```
## -------------|-----------|-----------|-----------|
##            1 |       350 |        58 |       408 |
##              |     2.934 |    46.023 |           |
##              |     0.858 |     0.142 |     0.085 |
##              |     0.077 |     0.201 |           |
##              |     0.073 |     0.012 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      4533 |       289 |      4822 |
##              |     0.940 |     0.060 |           |
## -------------|-----------|-----------|-----------|
##
##
```

For logistic regression, 14.2% of those who are predicted to purchuse do purchase.