Module 8: Recommended Exercises

Statistical Learning V2021

alexaoh

10 mars, 2021

Problem 1 – Theoretical

a) Provide a detailed explanation of the algorithm that is used to fit a regression tree. What is different for a classification tree?

Answer: Algorithm: The prediction space is split in such a way that a criterion function across all regions is smallest. This criterion function is different depending on if we are building a regression tree (RSS) or a classification tree (Gini-index or Cross-entropy). When this split into non-overlapping regions of the predictor space is done, we make the same prediction for each observation that falls into the same region - the mean of the responses for the training observations that fall into the region (regression tree) or some sort of majority vote (classification tree). How are these splits found? For a regression tree, we could try to minimize $\text{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, where $\hat{y}_{R_j}$ is the mean response for the training data in region j (and the predicted value for new observations that fall into region j). However, an exhaustive search over all partitions of the predictor space is computationally infeasible. Therefore, we use a greedy approach called Recursive Binary Splitting: We find a split in each step which minimizes RSS, where each single split in each step only depends on one of the predictors. In the next step, we split only one of the previously split regions. This algorithm is continued until the stopping criterion of choice is reached. The predictive performance of a tree can be improved by pruning, i.e. Cost Complexity Pruning. This is done by growing a very large tree and then pruning the tree back to obtain a subtree. We try to find a tree that minimizes $C_\alpha(T) = Q(T) + \alpha|T|$, where $Q(T)$ is our cost function and $|T|$ is the number of terminal nodes in subtree T. The parameter α penalizes larger trees, i.e. with more leaves. For regression trees we choose the RSS for the subtree, as defined above. as the cost function $Q(T)$. We can use K-fold cross-validation to find the optimal value of α.

The first difference for a classification tree is the criterion used when making binary splits. In the regression setting the RSS is used, since we have a quantitative response. On the contrary, when building a classification tree, the classification error rate could perhaps be used, i.e. the fraction of training observations that do not belong to the majority class in each region. However, this criterion is not sufficiently sensitive when building classification trees, which is the reason behind why two other criteria are used in practice: The Gini-index $G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$ or cross-entropy $D = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$. These two metrics are measures of node impurity, which we want to minimize in our tree. The Gini-index is a measure of the total variance across the K classes. Cross-entropy is defined differently, but is quite similar numerically to the Gini-index. Also, the Gini-index and Cross-entropy are differentiable, which may be useful in numerical optimization. Moreover, the second difference for a classification tree is that predictions in classification trees are done with a majority vote (in each region) or by estimation of the probability that the observation belongs to each class (proportion of points in each region that belong to each class), instead of the mean. Still, the class with the highest estimated probability will get the classification of the new observation.

b) What are the advantages and disadvantages of regression and classification trees?

Advantages: Interpretability (nice graphical display, when sufficiently small), closer mirror of human decision-making, easily explained concept, handling of qualitative predictors without dummy variables, automatically implements interactions, automatically selects variables.

Disadvantages: Generally has worse predictive accuracy compared to other classical methods (high variance). Hence, a small change in data may cause a large change in the final estimated tree.

c) What is the idea behind bagging and what is the role of bootstap? How do random forests improve that idea?

Answer: The idea behind bagging is to make use of several consecutive bootstrap samples to build many trees. on each of these samples. In the end, the predictions from each of these trees are averaged, in order to reduce the variance of the predictions. Random forests improve on that idea by restricting the amount of predictors that may be chosen by the algorithm when splitting the regions, i.e. when building the trees. In each split, a random selection of predictors may be used as options to produce the split (typically $\sqrt{p}$ predictors in classification anf $\frac{p}{3}$ predictors in regression). In this manner the trees become less correlated (since more of the trees are potentially different), which may lead to a further decrease in variance of the predictions.

d) What is an out-of bag (OOB) error estimator and what percentage of observations are included in an OOB sample? (Hint: The result from RecEx5-Problem 4c can be used)

Answer: An OOB error estimator is the average (for regression) or the majority vote (for classification) among the predicted response based on the trees where the given predictor is OOB, i.e. the observation was not used when building a tree using the bootstrap sample. About $\frac{1}{3}$ of the observations are included in the OOB-sample, because, as the result from RecEx5-Problem 4c shows, the probability that a given observation is in a bootstrap sample is $\approx \frac{2}{3}$. Hence, the observations that are left out of each bootstrap sample may be used as "testing" data on the tree that was built with that bootstrap sample. This means that for B bootstrap samples, observation i will be outside the bootstrap sample in $\approx \frac{B}{3}$ of the fitted trees. The out-of-bag error for observation i can be calculated by taking the average (regression) or the majority vote (classification) of all the $\approx \frac{B}{3}$ predictions on each tree.

e) Bagging and Random Forests typically improve the prediction accuracy of a single tree, but it can be difficult to interpret, for example in terms of understanding which predictors are how relevant. How can we evaluate the importance of the different predictors for these methods?

Answer: We can make *variable importance plots*, which show the relative importance of the different predictors when making predictions. In general, there are two different types of variable importance plots. The first it based on decrease in node impurity and the second is based on randomization.

Variable importance based on node impurity relates to total decrease in the node impurity over split for a predictor. For regression trees, the total amount the RSS is decreased due to splits for each predictor is recorded and averaged over all the trees used when bagging or in random forests. For classification trees, the importance is the mean decrease in the Gini-index by splits of a predictor, over all trees.

Variable importance based on randomization is calculated using the OOB sample. Computations are carried out for one bootstrap sample at a time. Each time a tree is grown, the OOB sample is used to test the predictive power of the tree. For one predictor at a time, the OOB observations are permuted and the new OOB error is calculated. A large increase in this error (a large decrease in predictive performance) suggests that the predictor is of importance. The difference between the OOB error before and after the permutation is averaged over all trees and normalized by the standard deviation of the differences, in order to produce the final variable importance ratings based on randomization.

Problem 2 – Regression (Book Ex. 8)

In the lab, a classification tree was applied to the Carseats data set after converting the variable `Sales` into a qualitative response variable. Now we will seek to predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

a) Split the data set into a training set and a test set. (Hint: Use 70% of the data as training set and the rest 30% as testing set)

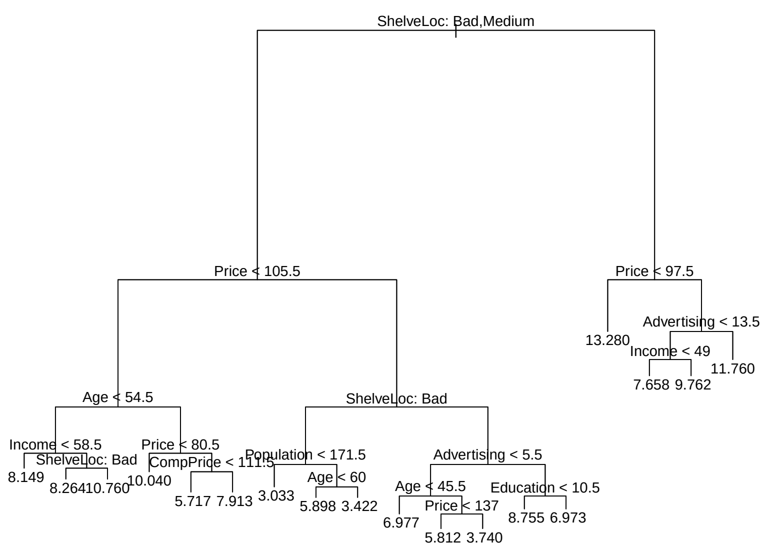
```r
library(ISLR)
data("Carseats")
set.seed(4268)
n = nrow(Carseats)
train = sample(1:n, 0.7 * n, replace = F)
test = (1:n)[-train]
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

b) Fit a regression tree to the training set using the default parameters for the stopping criterion. Plot the tree, and interpret the results. What test MSE do you obtain?

```r
library(tree)
tree.mod = tree(Sales ~ ., data = Carseats.train)
summary(tree.mod)
```

```
#>
#> Regression tree:
#> tree(formula = Sales ~ ., data = Carseats.train)
#> Variables actually used in tree construction:
#> [1] "ShelveLoc"   "Price"       "Age"         "Income"      "CompPrice"
#> [6] "Population"  "Advertising" "Education"
#> Number of terminal nodes:  18
#> Residual mean deviance:  2.609 = 683.6 / 262
#> Distribution of residuals:
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -3.74000 -1.12400 -0.06522  0.00000  1.06800  4.47200
```

```r
plot(tree.mod)
text(tree.mod, pretty = 0)
```




```r
# Calculate test MSE.
yhat <- predict(tree.mod, newdata = Carseats.test)
```

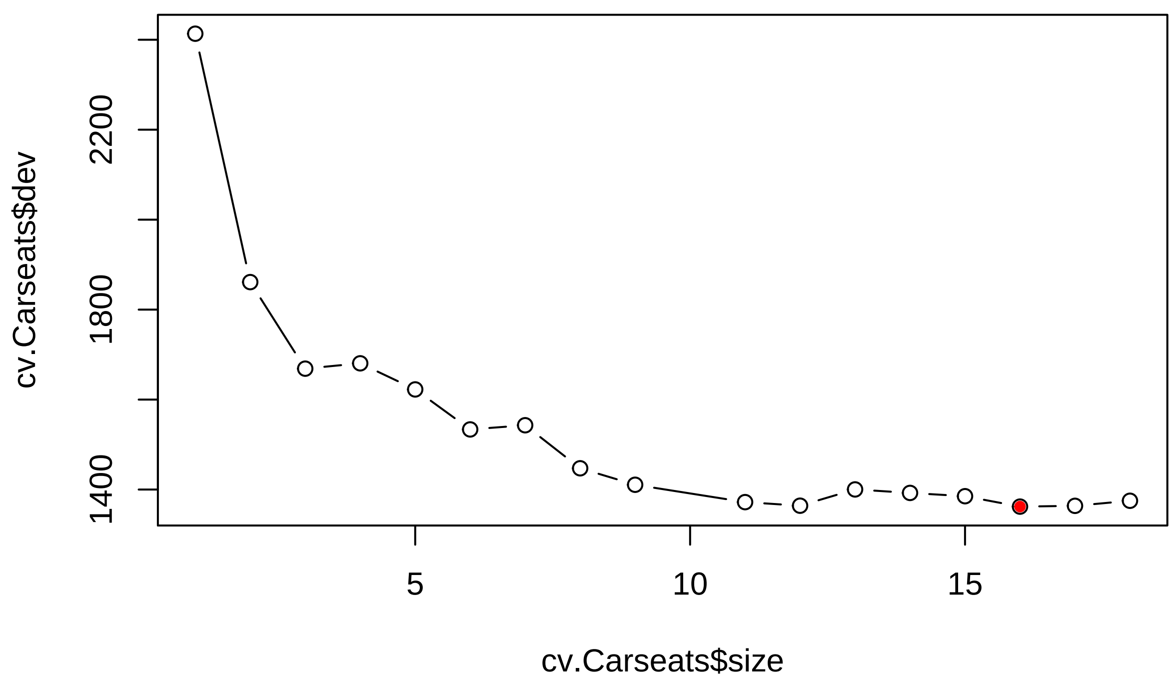
```r
mse <- mean((yhat - Carseats.test$Sales)^2)
mse
```

```
#> [1] 4.585249
```

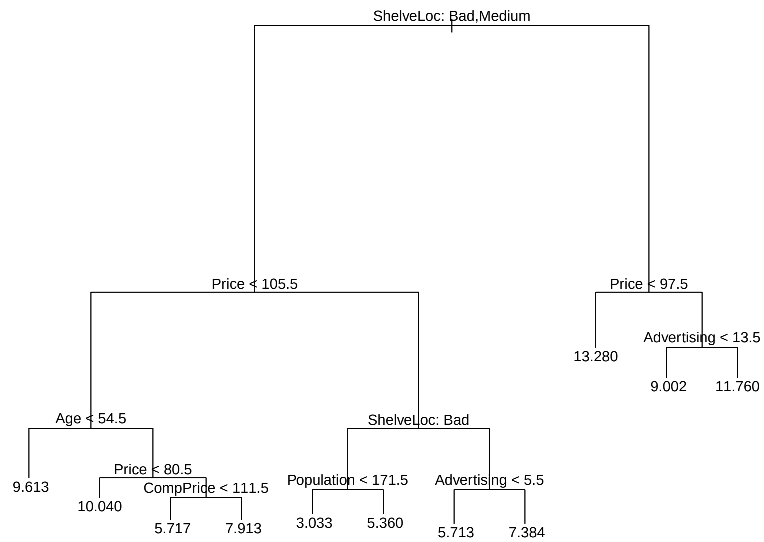
The results are hard to interpret in a hurry because the tree is relatively "bushy".

c) Use cross-validation in order to determine an optimal level of tree complexity. Does pruning the tree improve the test MSE?

```r
set.seed(4268)
cv.Carseats = cv.tree(tree.mod)
tree.min = which.min(cv.Carseats$dev)
best = cv.Carseats$size[tree.min]
plot(cv.Carseats$size, cv.Carseats$dev, type = "b")
points(cv.Carseats$size[tree.min], cv.Carseats$dev[tree.min], col = "red",
    pch = 20)
```




```r
# Despite best = 16, we choose best = 11, since it is smaller and
# almost as good.
k.best.tree <- prune.tree(tree.mod, best = 11)
plot(k.best.tree)
text(k.best.tree, pretty = 0)
```




```r
# Calculate test MSE.
yhat2 <- predict(k.best.tree, newdata = Carseats.test)
mse2 <- mean((yhat2 - Carseats.test$Sales)^2)
mse2
```

```
#> [1] 4.378499
```

Hence, pruning slightly improves the test MSE.

d) Use the bagging approach with 500 trees in order to analyze the data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important.

R-hints

```r
library(randomForest)
dim(Carseats)
```

```
#> [1] 400  11
```

```r
# mtry = 10 in order to consider all` predictors in each split -->
# bagging.
bag.Carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 10,
    ntree = 500, importance = TRUE)
yhat.bag <- predict(bag.Carseats, newdata = Carseats.test)
mse3 <- mean((yhat.bag - Carseats.test$Sales)^2)
mse3
```

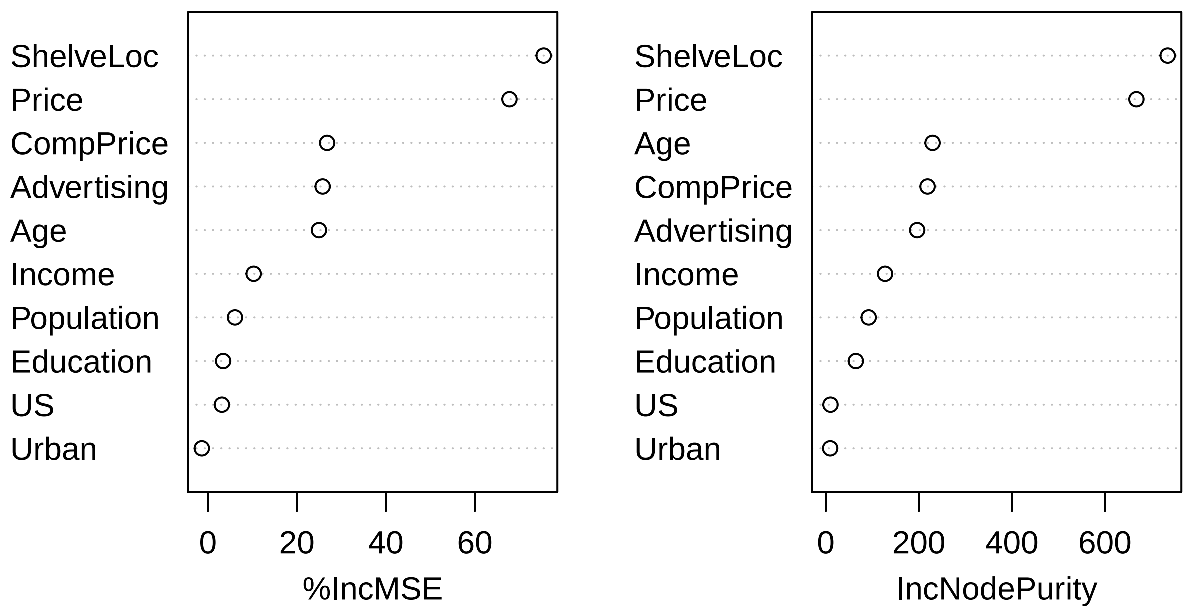
```
#> [1] 2.122958
```

```r
importance(bag.Carseats)
```

```
#>               %IncMSE IncNodePurity
#> CompPrice   26.803869    218.740455
#> Income      10.284817    127.447480
#> Advertising 25.795425    196.438893
#> Population   6.084270     92.149065
```

```
#> Price        67.791459    667.696518
#> ShelveLoc    75.485534    734.902022
#> Age          24.961130    229.491494
#> Education     3.423565     64.510742
#> Urban        -1.373635      9.423406
#> US            3.141449     10.105870
```

```
varImpPlot(bag.Carseats, main = "")
```



It is apparent that the test MSE is reduced quite a bit. `ShelveLoc` and `Price` give the largest decrease in node impurity and in MSE (both the variable imporance metrics agree on these two predictors). The rest of the variables give rise to slight disagreement between the two metrics.

e) Use random forests and to analyze the data. Include 500 trees and select 3 variables for each split. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
rf.Carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 3,
    ntree = 500, importance = TRUE)
yhat.rf <- predict(rf.Carseats, newdata = Carseats.test)
mse4 <- mean((yhat.rf - Carseats.test$Sales)^2)
mse4
```

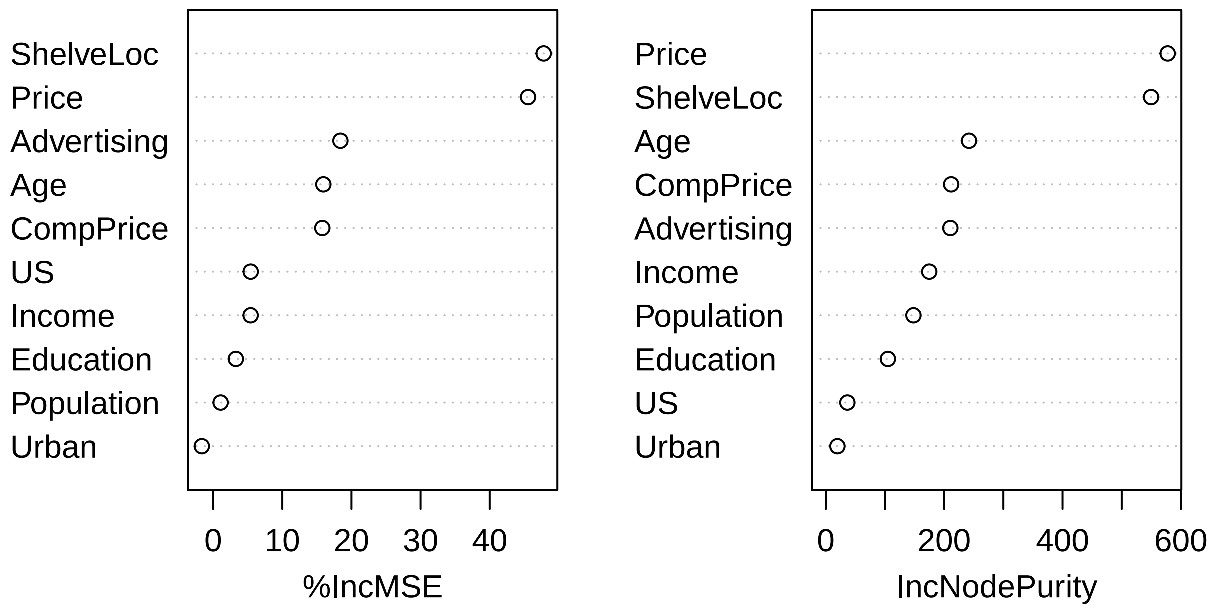
```
#> [1] 2.25397
```

```
importance(rf.Carseats)
```

```
#>                 %IncMSE IncNodePurity
```

```
#> CompPrice   15.789484     211.79213
#> Income       5.415374     174.79625
#> Advertising 18.402600     210.47149
#> Population   1.076874     148.09993
#> Price       45.548596     577.68865
#> ShelveLoc   47.810006     549.62278
#> Age         15.936114     241.99130
#> Education    3.275725     104.89503
#> Urban       -1.646580      19.63668
#> US           5.427599      36.45647
```

```r
varImpPlot(rf.Carseats, main = "")
```



The test MSE for a random forest with $m = 3$ is slightly higher than that of bagging, but it still lower than for pruning one tree.

f) Finally use boosting with 500 trees, an interaction depth $d = 4$ and a shrinkage factor $\lambda = 0.1$ (default in the `gbm()` function) on our data. Compare the MSE to all other methods.

```r
library(gbm)
r.boost = gbm(Sales ~ ., Carseats.train, distribution = "gaussian", n.trees = 500,
    interaction.depth = 4, shrinkage = 0.1)
yhat.r.boost <- predict(r.boost, newdata = Carseats.test, n.trees = 500)
mse5 <- mean((yhat.r.boost - Carseats.test$Sales)^2)
mse5
```

```
#> [1] 2.151292
```

The test MSE is further decreased when using boosting.

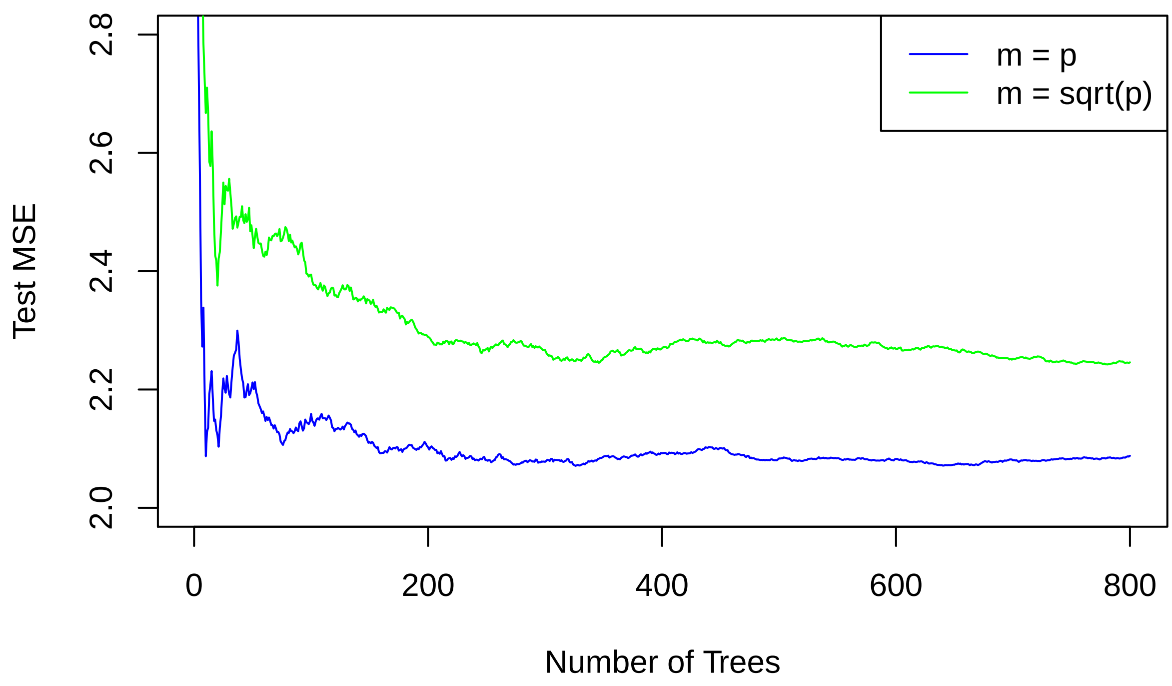
g) What is the effect of the number of trees (`ntree`) on the test error? Plot the test MSE as a function of `ntree` for both the bagging and the random forest method.

```
# Remove Sales from predictors.
train.predictors <- Carseats.train[, -1]
test.predictors <- Carseats.test[, -1]

# Make list of responses.
Y.train <- Carseats.train[, 1]
Y.test <- Carseats.test[, 1]

bag.Car <- randomForest(train.predictors, y = Y.train, xtest = test.predictors,
    ytest = Y.test, mtry = 10, ntree = 800)
rf.Car <- randomForest(train.predictors, y = Y.train, xtest = test.predictors,
    ytest = Y.test, mtry = 3, ntree = 800)

plot(1:800, bag.Car$test$mse, col = "blue", type = "l", xlab = "Number of Trees",
    ylab = "Test MSE", ylim = c(2, 2.8))
lines(1:800, rf.Car$test$mse, col = "green")
legend("topright", c("m = p", "m = sqrt(p)"), col = c("blue", "green"),
    cex = 1, lty = 1)
```



We can see that $B = 500$ seems to be a reasonable choice, since the testMSE is relatively stable from thereon out.

Problem 3 – Classification

In this exercise you are going to implement a spam filter for e-mails by using tree-based methods. Data from 4601 e-mails are collected and can be uploaded from the kernlab library as follows:

```
library(kernlab)
data(spam)
dim(spam)
```

```
#> [1] 4601   58
```

Each e-mail is classified by `type` (`spam` or `nonspam`), and this will be the response in our model. In addition there are 57 predictors in the dataset. The predictors describe the frequency of different words in the e-mails and orthography (capitalization, spelling, punctuation and so on).

a) Study the dataset by writing `?spam` in R.

b) Create a training set and a test set for the dataset. (Hint: Use 70% of the data as training set and the rest 30% as testing set)

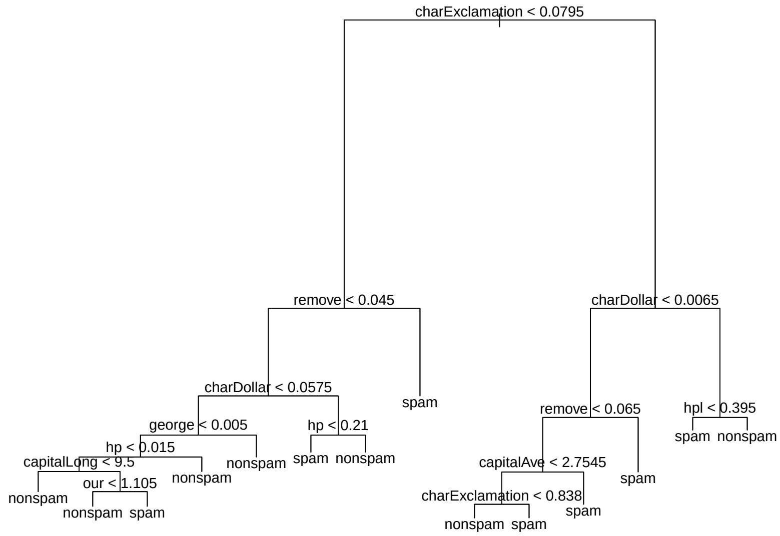
```
set.seed(4268)
n = nrow(spam)
train = sample(1:n, 0.7 * n, replace = F)
test = (1:n)[-train]
spam.train = spam[train, ]
spam.test = spam[-train, ]
```

c) Fit a tree to the training data with `type` as the response and the rest of the variables as predictors. Study the results by using the `summary()` function. Also create a plot of the tree. How many terminal nodes does it have?

```
tree.spam <- tree(type ~ ., data = spam.train, split = "deviance")  # Using cross entropy as criterion.
summary(tree.spam)
```

```
#>
#> Classification tree:
#> tree(formula = type ~ ., data = spam.train, split = "deviance")
#> Variables actually used in tree construction:
#> [1] "charExclamation" "remove"          "charDollar"      "george"
#> [5] "hp"              "capitalLong"     "our"             "capitalAve"
#> [9] "hpl"
#> Number of terminal nodes:  14
#> Residual mean deviance:  0.4801 = 1539 / 3206
#> Misclassification error rate: 0.08975 = 289 / 3220
```

```
plot(tree.spam)
text(tree.spam, pretty = 0)
```



The tree has 14 terminal nodes.

d) Predict the response on the test data. What is the misclassification rate?

```
library(caret)
yhat.spam <- predict(tree.spam, newdata = spam.test, type = "class")

confMat <- confusionMatrix(yhat.spam, reference = spam.test$type)$table
confMat
```

```
#>           Reference
#> Prediction nonspam spam
#>    nonspam     781   67
#>    spam         67  466
```

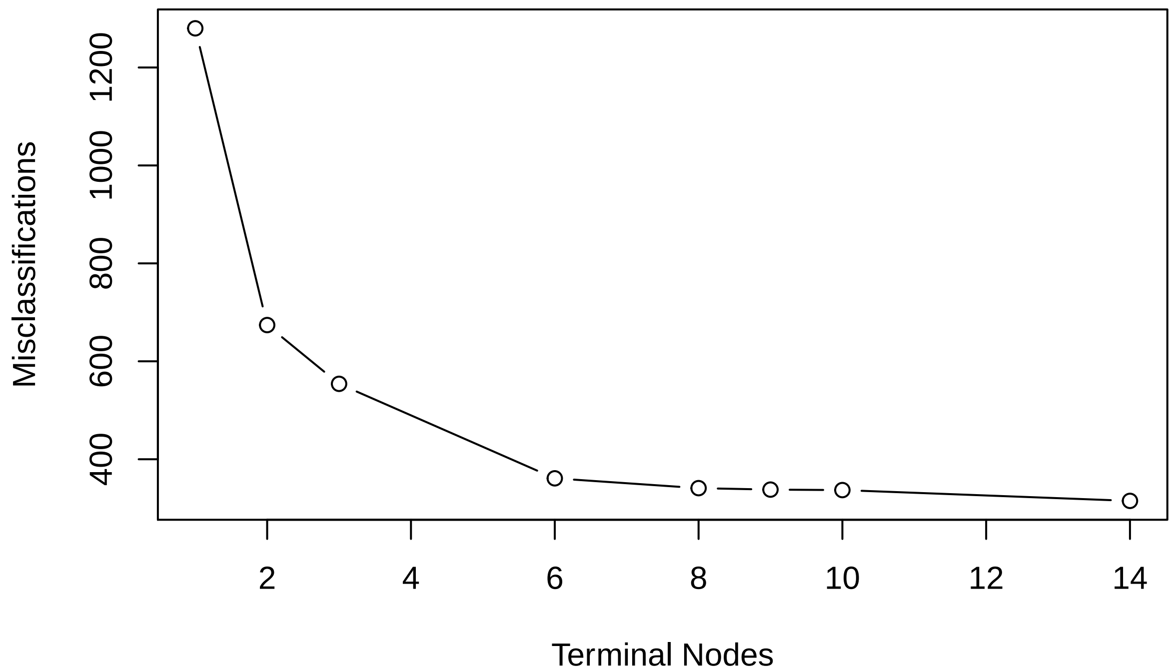
```
# This produces the same as the table above. matrix <-
# table(yhat.spam, spam.test$type) matrix

misclass.rate <- 1 - sum(diag(confMat))/sum(confMat[1:2, 1:2])
misclass.rate
```

```
#> [1] 0.09703114
```

e) Use the `cv.tree()` function to find an optimal tree size. Prune the tree according to the optimal tree size by using the `prune.misclass()` function and plot the result. Predict the response on the test data by using the pruned tree. What is the misclassification rate in this case?

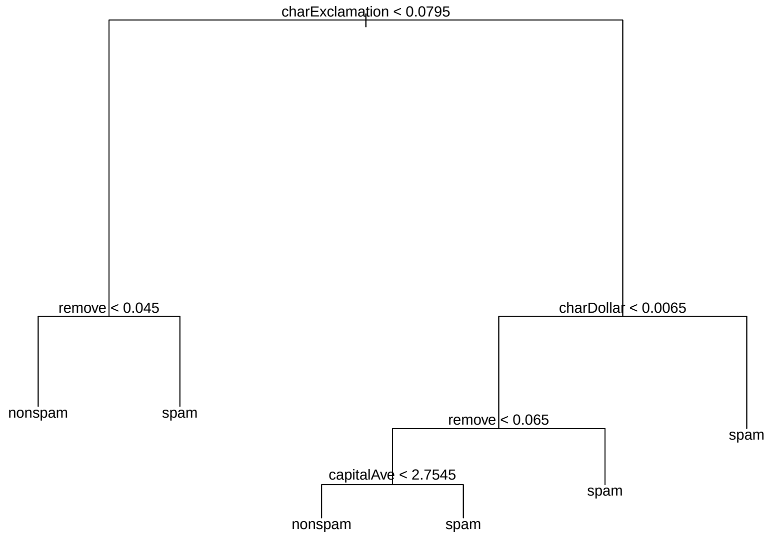
```
cv.spam <- cv.tree(tree.spam, FUN = prune.misclass)
plot(cv.spam$size, cv.spam$dev, type = "b", xlab = "Terminal Nodes",
    ylab = "Misclassifications")
```




```
data.frame(cv.spam$size, cv.spam$dev)  # Will use size = 8 based on this.
```

cv.spam.size	cv.spam.dev
14	315
10	337
9	338
8	341
6	361
3	554
2	674
1	1280

```
pruned.spam <- prune.misclass(tree.spam, best = 6)
plot(pruned.spam)
text(pruned.spam)
```




```
yhat.pruned.spam <- predict(pruned.spam, newdata = spam.test, type = "class")

confmatrix <- table(yhat.pruned.spam, spam.test$type)
confmatrix
```

```
#>
#> yhat.pruned.spam nonspam spam
#>          nonspam     796  104
#>          spam         52  429
```

```
misclass.rate2 <- 1 - sum(diag(confmatrix))/sum(confmatrix)
misclass.rate2
```

```
#> [1] 0.1129616
```

The misclassification rate is higher for the pruned tree with 8 terminal nodes.

f) Create a decision tree by using the bagging approach with $B = 500$. Use the function `randomForest()` and consider all of the predictors in each split. Predict the response on the test data and report the misclassification rate.

```
dim(spam)
```

```
#> [1] 4601   58
```

```
bag.spam = randomForest(type ~ ., data = spam.train, mtry = 57, ntree = 500,
    importance = TRUE)
yhat.bag.spam <- predict(bag.spam, newdata = spam.test)

confmatrix2 <- table(yhat.bag.spam, spam.test$type)
confmatrix2
```

```
#>
#> yhat.bag.spam nonspam spam
#>       nonspam     811   42
```

```
#>      spam         37  491
```

```
misclass.rate3 <- 1 - sum(diag(confmatrix2))/sum(confmatrix2)
misclass.rate3
```

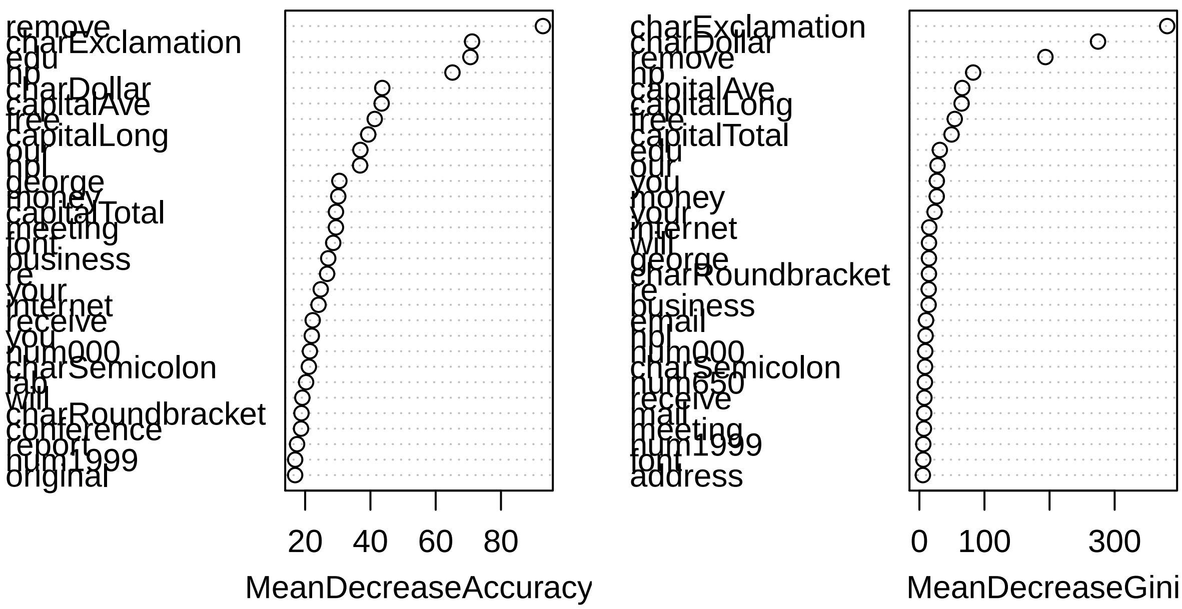
```
#> [1] 0.05720492
```

```
importance(bag.spam)
```

```
#>                   nonspam       spam MeanDecreaseAccuracy MeanDecreaseGini
#> make           1.15095110  5.4497708            4.9897985       4.62755236
#> address       11.61825673  8.8548321           14.3146698       5.36444015
#> all            1.92683004  4.9348402            5.0986082       4.77435295
#> num3d         10.57209854  2.8435952           10.3160595       2.21860836
#> our           24.90710250 32.6005029           36.9235277      27.87876348
#> over          12.07248095  1.9715450           11.8417583       4.15159336
#> remove        82.02270313 51.2890325           92.8574007     193.62969183
#> internet      22.20835624 10.3123698           24.0906913      15.21448643
#> order         12.47776818  4.0944528           12.9420271       3.39056617
#> mail           9.65496981  6.2310668           11.8500531       7.47279467
#> receive       21.23872523  5.5869088           22.3155835       7.71442695
#> will           5.27692488 19.9389045           19.1412359      14.78754152
#> people         0.81012545  7.1267917            6.0328221       4.48138804
#> report        12.19373277 13.2128200           17.5199022       5.33829771
#> addresses      5.13377878  4.4185240            6.0085277       0.97177730
#> free          32.92219935 27.9415037           41.3195028      54.31127407
#> business      26.28143139  7.4330687           27.0702188      14.18741462
#> email         14.74243951  5.1094882           14.9255294      10.27188415
#> you           14.66271525 17.3719688           22.0338723      26.79652724
#> credit         5.04667976  1.8658667            5.5235321       1.15075374
#> your          19.52671504 15.4976759           24.7738065      23.33336902
#> font          26.45372248 19.1120419           28.6014081       5.98094629
#> num000        21.22484733 -0.5034415           21.4549458       8.97789743
#> money         28.18834888 14.7163465           30.1428812      26.65900431
#> hp            24.61095260 68.4725034           65.1392862      82.64217996
#> hpl            0.06637866 36.8572701           36.8381793       9.56800688
#> george        10.70607811 28.5003585           30.4889495      14.70294631
#> num650        10.91828335 10.6606641           15.4907626       8.49452529
#> lab           -6.49948799 20.2902362           20.2883045       2.35872342
#> labs           6.04685776  2.5926280            6.5651844       2.08572752
#> telnet        -0.48394562  8.3562670            8.6230518       0.72322032
#> num857        -3.57687627 11.7771331           11.7788173       0.93293103
#> data          -0.61056279  6.5781366            5.3764310       3.03622874
#> num415         1.42108345  3.8475585            4.2548950       0.42422067
#> num85          2.33975444  8.6330493            9.0158604       1.61177342
#> technology    13.55582774  4.6679339           13.7934170       4.56428493
#> num1999        5.44193096 17.0110438           16.9340052       5.98837384
#> parts         -2.68871848  6.3420020            1.6601161       0.79945176
#> pm            -0.19230859 12.7754327           12.1416553       3.84378208
#> direct         6.99399903 -1.8172348            6.9394766       0.89684472
#> cs             0.32420482  2.3997766            2.4222249       0.23928684
#> meeting        7.66372235 29.4375336           29.4360980       6.97450790
#> original      -2.38553950 16.9918670           16.9216839       3.25448521
#> project       -0.41702612 10.2607356            9.5844125       2.39039459
#> re            18.44329003 20.8264974           26.7235036      14.31754464
#> edu           19.32498315 71.1612123           70.6324668      31.41342902
```

```
#> table              -1.04410061  1.0010015          -0.8455594        0.09523428
#> conference          3.70042233 18.7715459          18.7430849        2.83362374
#> charSemicolon      16.53331358 13.8680254          21.1347450        8.71245648
#> charRoundbracket    7.60199259 19.2309805          18.8694097       14.69168396
#> charSquarebracket   2.97500412  2.4906659           3.9999396        1.20749580
#> charExclamation    54.68465353 52.1805189          71.1492987      380.70548489
#> charDollar         33.45736303 31.7676249          43.6329556      274.46385140
#> charHash            4.56233934  2.5323213           4.9130594        2.28485340
#> capitalAve         37.47488611 25.1051278          43.4404226       65.88331647
#> capitalLong        21.60718638 31.7068814          39.3247901       64.87330649
#> capitalTotal       25.23858437 15.0694370          29.4380943       49.38807762
```

```r
varImpPlot(bag.spam, main = "")
```



The misclassification rate is the lowest thus far among the methods used.

g) Apply the `randomForest()` function again with 500 trees, but this time consider only a subset of the predictors in each split. This corresponds to the random forest-algorithm. Study the importance of each variable by using the function `importance()`. Are the results as expected based on earlier results? Again, predict the response for the test data and report the misclassification rate.

```r
# Using \sqrt(57) \approx 7.5 \approx 8 predicotrs in each split.
rf.spam = randomForest(type ~ ., data = spam.train, mtry = 8, ntree = 500,
    importance = TRUE)
yhat.rf.spam <- predict(rf.spam, newdata = spam.test)

confmatrix3 <- table(yhat.rf.spam, spam.test$type)
confmatrix3
```

```
#>
#> yhat.rf.spam nonspam spam
#>      nonspam     820   39
#>      spam         28  494
```

```r
misclass.rate4 <- 1 - sum(diag(confmatrix3))/sum(confmatrix3)
misclass.rate4
```

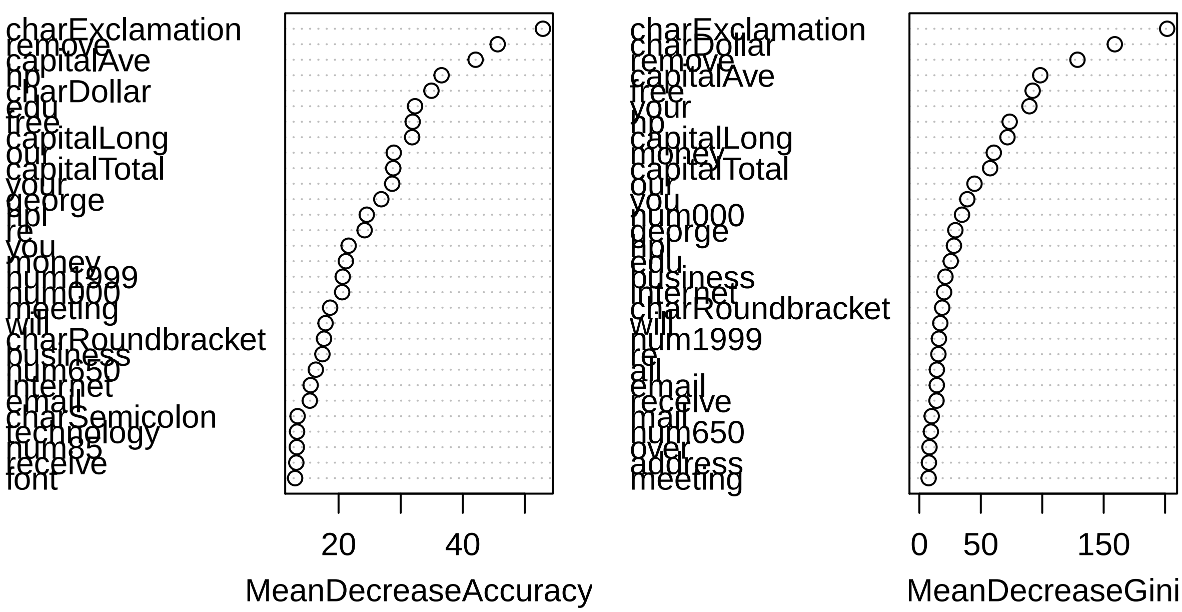
```
#> [1] 0.04851557
```

```r
importance(rf.spam)
```

```
#>                        nonspam        spam MeanDecreaseAccuracy MeanDecreaseGini
#> make                4.83264649  6.11499648          7.588782857        5.8734633
#> address             8.03742804  5.41963289         10.037871168        7.7640783
#> all                 3.45522984 11.94031414         12.155462522       14.2277686
#> num3d               3.81034171  1.43901873          4.126937857        0.9225658
#> our                21.46376498 23.70523154         28.890896445       44.8667685
#> over                9.56992113  8.84745516         11.853086543        8.2356398
#> remove             42.63176720 31.12184665         45.623034629      128.6888167
#> internet           15.14521346  9.71663999         15.502429398       20.1354418
#> order               9.03707166  5.50091018          9.004701889        6.0527363
#> mail                8.27867753  8.48321455         11.157122297       10.0286556
#> receive            13.44316221  4.81101597         13.200198439       13.9539274
#> will                6.54750488 17.56968237         17.902695764       17.0233122
#> people              0.02543218  7.13976413          6.330287576        4.8892994
#> report              6.33626225  7.29094682          8.980244916        3.7928751
#> addresses           6.84724156  4.65451329          7.799269891        2.2016251
#> free               28.92490312 23.27086624         31.935889806       92.2161257
#> business           16.42137537 10.77432830         17.399620407       21.2335005
#> email              12.49259638 10.89173836         15.370743373       14.1831034
#> you                13.01992702 18.39332179         21.618010457       39.0112450
#> credit              7.84449629  3.73380984          8.114071685        4.0904636
#> your               20.42039068 24.28265743         28.638495585       89.5738059
#> font               11.52089542  9.56052900         13.002970959        4.1634127
#> num000             19.89399578  9.32875837         20.580886885       34.7178710
#> money              18.59689946 15.23478904         21.176647995       60.5713442
#> hp                 24.15253840 35.05203740         36.586861695       73.4943107
#> hpl                14.13846530 22.18241219         24.542882977       28.1354989
#> george             17.28245054 24.37898272         26.880702760       29.2860949
#> num650             10.00309723 13.52816200         16.335239963        9.3496166
#> lab                 1.31848044  9.46175967          9.805171864        2.5986628
#> labs                4.39006995 10.16411860         11.238046400        6.3074342
#> telnet              4.05925222  8.08623222          8.800939354        2.8903015
#> num857              2.39240276  6.54810067          6.826259164        0.9706725
#> data                3.94114460  9.00298604         10.164646242        4.1884451
#> num415              3.60910041  6.76879734          7.598122618        1.2406945
#> num85               5.14682086 12.63464958         13.273576911        5.3388890
#> technology          9.63902738  9.25606875         13.318946822        5.1332471
#> num1999            13.89208331 18.17016435         20.676021841       15.8866750
#> parts              -1.74301919  2.97848276         -0.003215885        0.6234623
#> pm                  4.40321219 11.16285719         11.606261502        4.8834063
#> direct              6.46055792  2.85299535          7.075783004        1.9790784
#> cs                  1.47546166  6.42725801          6.544265445        1.1091305
#> meeting            10.65581179 17.10097080         18.643573669        7.5354974
#> original           -0.00948209 10.84815595         10.776115413        2.5149200
```

```
#> project              4.05714553  8.85099749           9.324875617         3.0957500
#> re                  16.33159076 18.79607320          24.194268241        15.4900349
#> edu                 20.78627783 29.97418819          32.317541625        25.5194418
#> table               -0.65194914  0.04918159          -0.525283265         0.4057492
#> conference           2.39962258  8.75774978           8.663384490         1.9889401
#> charSemicolon       10.60546670  8.88526228          13.398801399         7.0184387
#> charRoundbracket     7.18305278 18.50370081          17.656606685        18.5817122
#> charSquarebracket    7.62492720  6.52214268           9.371143591         3.9415461
#> charExclamation     38.51039230 45.16650890          52.913128390       201.6978017
#> charDollar          28.79576044 27.71113288          34.956049222       159.0541825
#> charHash             5.84420827  5.02646612           7.656421601         4.9181391
#> capitalAve          31.34082075 28.53475765          42.072926446        98.3957821
#> capitalLong         23.49891946 22.64873030          31.850206175        71.7139370
#> capitalTotal        22.74736925 20.11457113          28.806846164        57.5953200
```

```r
varImpPlot(rf.spam, main = "")
```



The misclassification rate is, again, the lowest thus far among the methods used (however, the decrease is not large compared to the bagging results). This is as expected, since random forests reduce the correlation between the trees.

h) Use `gbm()` to construct a boosted classification tree using 5000 trees, an interaction depth of $d = 3$ and a shrinkage parameter of $\lambda = 0.001$. Predict the response for the test data and report the misclassification rate.

```r
# The Bernoulli distribution of gbm only permits 0/1.
spamboost <- spam
spamboost$type <- c()
```

```r
spamboost$type[spam$type == "spam"] <- 1
spamboost$type[spam$type == "nonspam"] <- 0

spam.boost = gbm(type ~ ., spamboost[train, ], distribution = "bernoulli",
    n.trees = 5000, interaction.depth = 3, shrinkage = 0.001)

yhat.spam.boost <- predict(spam.boost, newdata = spamboost[-train, ],
    n.trees = 5000, distribution = "bernoulli", type = "response")

yhat.spam.boost <- ifelse(yhat.spam.boost > 0.5, 1, 0)  # Transform probabilities to 0/1.
confmatrix4 <- table(yhat.spam.boost, spam.test$type)
confmatrix4
```

```
#>
#> yhat.spam.boost nonspam spam
#>               0     812   52
#>               1      36  481
```

```r
misclass.rate5 <- 1 - sum(diag(confmatrix4))/sum(confmatrix4)
misclass.rate5
```

```
#> [1] 0.06372194
```

i) Compare the misclassification rates in d-h. Which method gives the lowest misclassification rate for the test data? Are the results as expected?

We get lower misclassification rates for the test data for bagging, random forests and boosting, compared to pruning, which is as expected. Furthermore, all methods seem to agree that the three most important predictors are `charExclamation`, `remove` and `charDollar`.