­

Ch8. Tree-Based Methods

-Lab & Exercises #8,9,10

­­

**학번 : 182STG12**

**이름 : 오혜윤**

c

Problem Description

In XGboost tutorial, we learn about the xgboost() function. There are several useful parameters we can choose in xgboost() function. We first split the dataset into the training set and the test set. Then, we choose the classification model and the tuning parameters. We can decrease the test error as we choose the right tuning parameters.

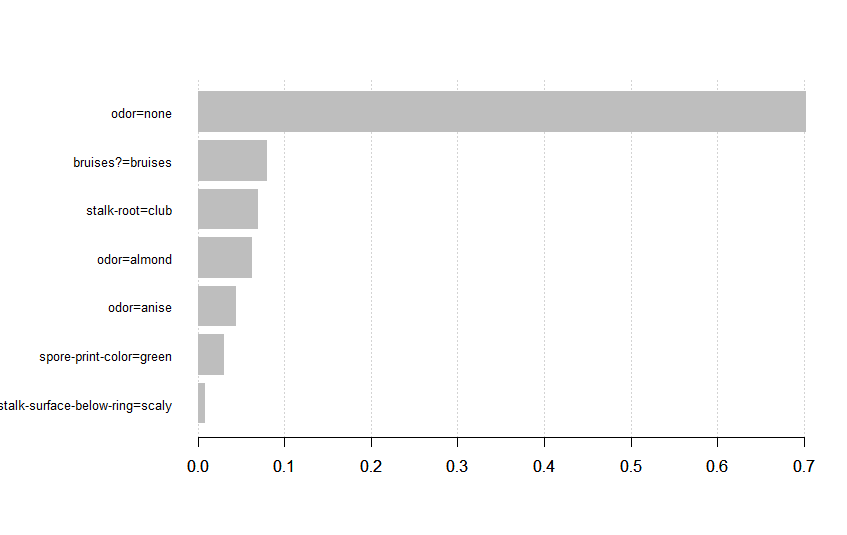
In Chapter 8 lab, we learn about trees, bagging and boosting. Using tree-based methods, we fit the model and compare the test errors. There are tree-based methods such as regression tree, bagging, random forests and boosting. If we want to choose the number of components and consider what the most important variable is, we can use cv.tree() function and importance() function ,respectively. Comparing these tree-based methods, we can choose the model with low test error.

In exercises 8, 9, and 10, using CarSeats, OJ and Hitters dataset, we apply various tree-based methods. We split the training set and the test dataset. Then, we fit the models and calculate the test errors and cv errors. Comparing the errors, we can find the best tree model with the optimal number of components. Also, we can find the most important variable which affects y.

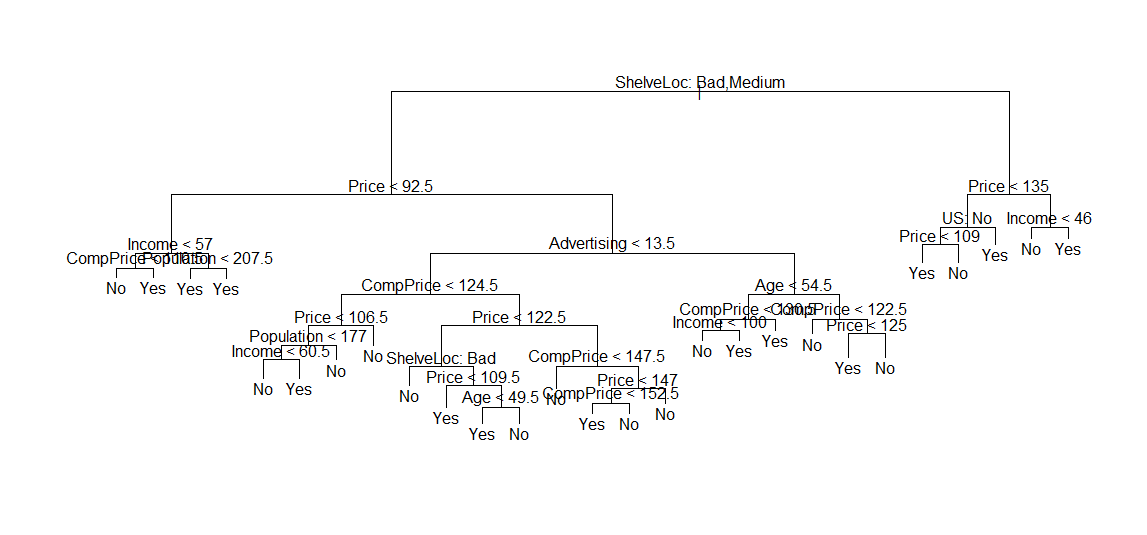
Results

**XGboost Tutorial Review**

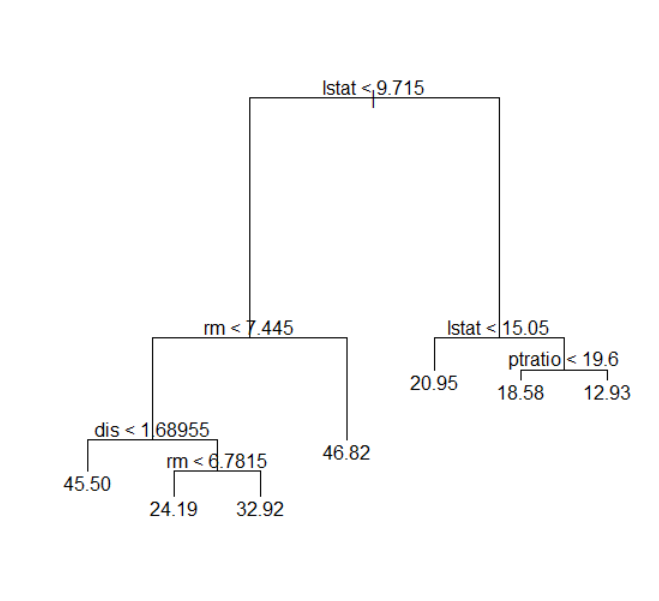
In this xgboost package tutorial, we can learn about the function xgboost(). First, we split dataset into the training set and the test set. Then, we change the tuning parameters to choose the right xgboost model for dataset. The plot below represents the importance of variables. We can load or save the model using xgb.save() and xgb.load().



**CH8. Lab Review**

In this lab, it was mostly about trees. First, using Carseats dataset, we learn about classification trees to classify the data. Then, we check the test error rates to evaluate the performance of a classification tree on these data. The test correct rate was about 71.5%.

After that, we determine the optimal level of tree complexity using function cv.tree() which performs cross-validation. Then, we do the same steps and estimate the test error rate. Using the pruned tree, the test correct rate was about 77% and we can conclude that using a pruned tree is better.

Second, using Boston dataset, we learn about regression trees. We create a training set and fit the tree to the training data. In this case, the most complex tree is selected by cross-validation so we use the unpruned tree to make predictions.

As a result, the test set MSE associated with the regression tree is 25.05. The square root of the MSE is therefore around 5.005, indicating that this model leads to test predictions that are within around $5, 005 of the true median home value for the suburb.

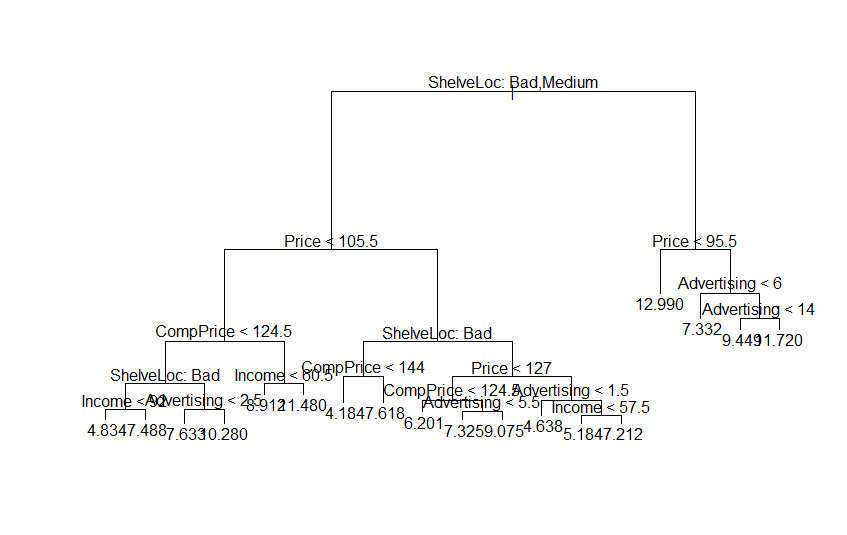
Third, we apply bagging and random forests to the Boston data, using the randomForest package in R. Using the randomForest() function, we perform both random forests and bagging. The test set MSE was about 13.16.

Last, using the gbm package, we do boosting. Using the gbm function, we fit the boosted regression trees to the Boston dataset. We can check the most important variables which affects the house prices a lot. We can also produce partial dependence plots for these variables. It will illustrate the marginal effect of the selected variables on the response after integrating out the other variables.

**CH8 Exercises**

1. **In the lab, a classification tree was applied to the Carseats data set after converting 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.**
2. **Split the data set into a training set and a test set.**

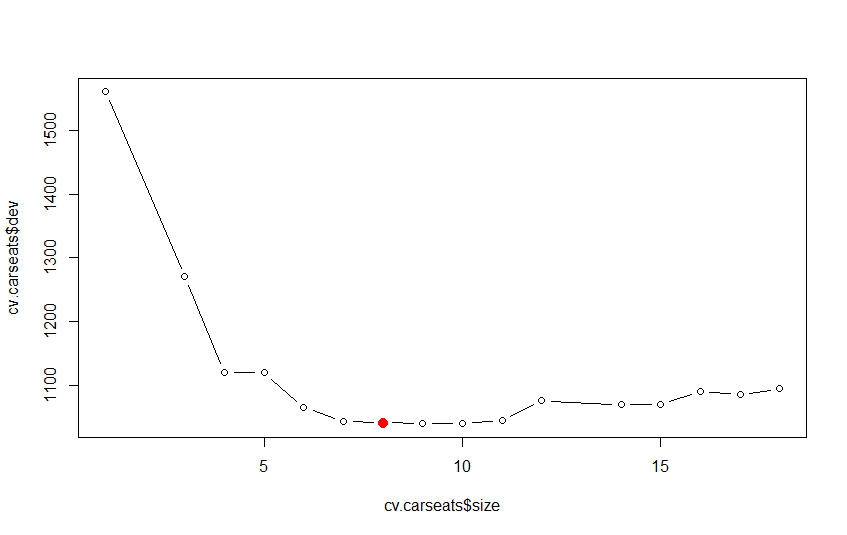
|  |
| --- |
| R codes: |
| >library(ISLR); set.seed(1)  >train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)  >Carseats.train <- Carseats[train, ]; Carseats.test <- Carseats[-train, ] |

1. **Fit a regression tree to the training set. Plot the tree, and interpret the results. What test error rate do you obtain?**

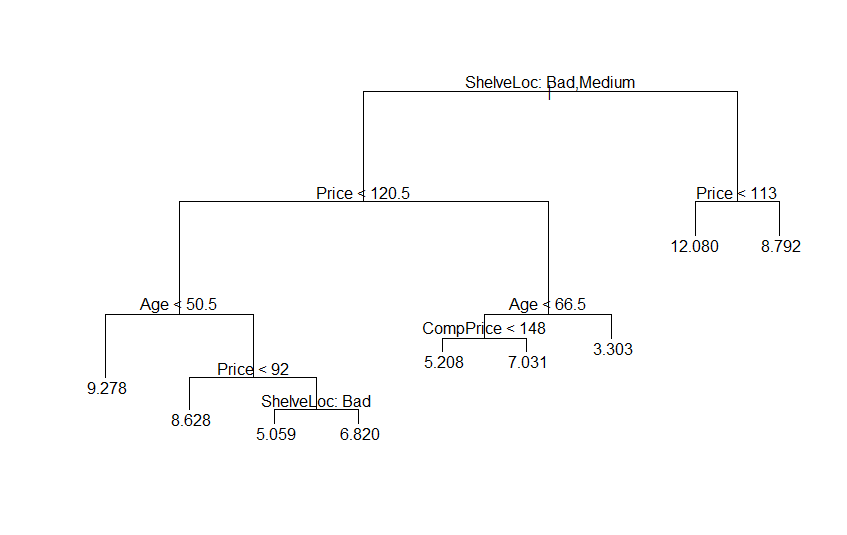
|  |
| --- |
| Results: |
| Regression tree:  tree(formula = Sales ~ ., data = Carseats.train)  Variables actually used in tree construction:  [1] "ShelveLoc" "Price" "Age" "Advertising" "Income"  [6] "CompPrice"  Number of terminal nodes: 18  Residual mean deviance: 2.36 = 429.5 / 182  Distribution of residuals:  Min. 1st Qu. Median Mean 3rd Qu. Max.  -4.2570 -1.0360 0.1024 0.0000 0.9301 3.9130 |

The regression tree above contains 18 terminal nodes. The order of importance of variables is ShelveLoc, Price, Age , Advertising, Income and CompPrice. Therefore, the most importance variable which affects Sales is “ShelveLoc”. The test error rate is 4.1489.

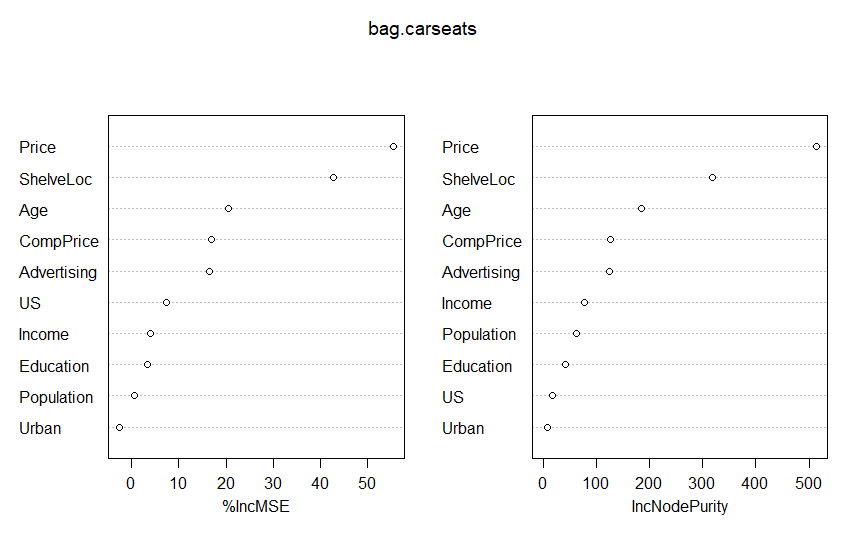
1. **Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?**



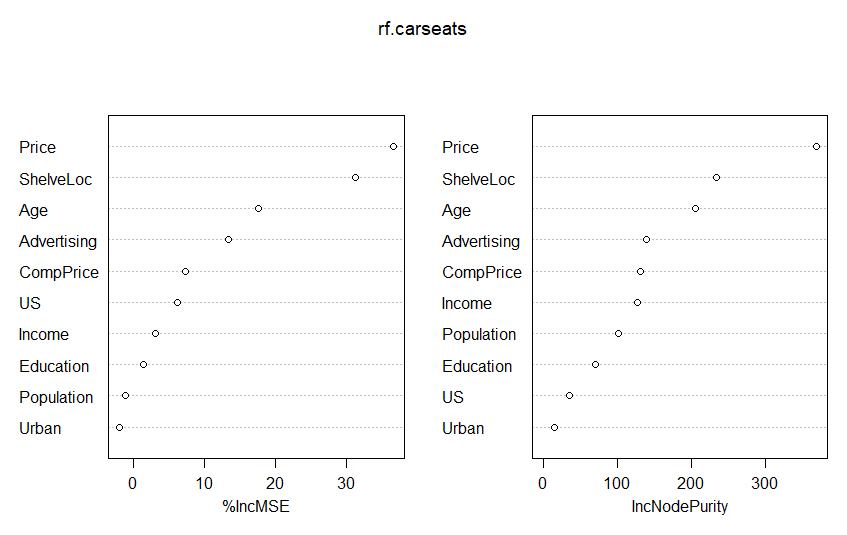
The optimal level of tree complexity is 8. Therefore, we now prune the tree to obtain the 8-node tree.



The left plot is the pruned tree plot which contains 8 nodes. The test error rate of the pruned tree is 4.9931. The test error rate of the pruned tree is a little bit higher(about 0.8) than that of the unpruned tree.

1. **Use the bagging approach in order to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important.**

The test error rate of the bagging model is 2.6339 which is the lowest MSE. The plot above represents the importance of the variables. If its values of IncMSE and IncNodePurity are high, then the variable is important. So, we can conclude that “Price” and “ShelveLoc” are the two most important variables.

1. **Use random forests to analyze this data. What test error rate 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.**

In this case, with m=sqrt(p), the test error rate is 3.3212. The plot above represents the importance of variables. We can conclude that “Price” and “ShelveLoc” are the two most important variables.

1. **This problem involves the OJ data set which is part of the ISLR package.**
   1. **Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.**

|  |
| --- |
| R codes: |
| >set.seed(1)  >train <- sample(1:nrow(OJ), 800)  > OJ.train <- OJ[train, ];OJ.test <- OJ[-train, ] |

* 1. **Fit a tree to the training data, with Purchase as the response and the other variables except for Buy as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?**

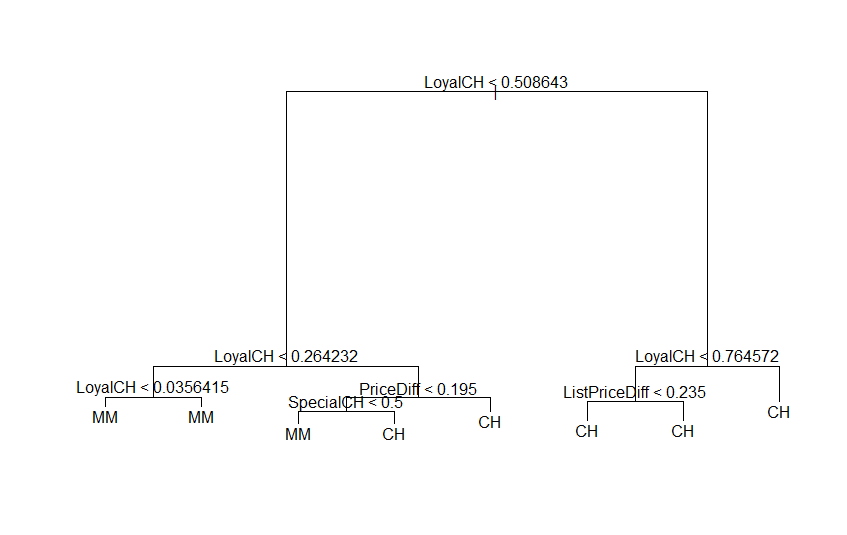
|  |
| --- |
| Results: |
| Classification tree:  tree(formula = Purchase ~ ., data = OJ.train)  Variables actually used in tree construction:  [1] "LoyalCH" "PriceDiff" "SpecialCH" "ListPriceDiff"  Number of terminal nodes: 8  Residual mean deviance: 0.7305 = 578.6 / 792  Misclassification error rate: 0.165 = 132 / 800 |

The training error rate is 0.165 and the tree has 8 terminal nodes.

* 1. **Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.**

We pick the node labelled 8, which is a terminal node because of the asterisk. The split criterion is LoyalCH < 0.035, the number of observations in that branch is 57 with a deviance of 10.07 and an overall prediction for the branch of MM. Less than 2% of the observations in that branch take the value of CH, and the remaining 98% take the value of MM.

|  |
| --- |
| Results: |
| node), split, n, deviance, yval, (yprob)  \* denotes terminal node  1) root 800 1064.00 CH ( 0.61750 0.38250 )  2) LoyalCH < 0.508643 350 409.30 MM ( 0.27143 0.72857 )  4) LoyalCH < 0.264232 166 122.10 MM ( 0.12048 0.87952 )  8) LoyalCH < 0.0356415 57 10.07 MM ( 0.01754 0.98246 ) \*  9) LoyalCH > 0.0356415 109 100.90 MM ( 0.17431 0.82569 ) \*  5) LoyalCH > 0.264232 184 248.80 MM ( 0.40761 0.59239 )  10) PriceDiff < 0.195 83 91.66 MM ( 0.24096 0.75904 )  20) SpecialCH < 0.5 70 60.89 MM ( 0.15714 0.84286 ) \*  21) SpecialCH > 0.5 13 16.05 CH ( 0.69231 0.30769 ) \*  11) PriceDiff > 0.195 101 139.20 CH ( 0.54455 0.45545 ) \*  3) LoyalCH > 0.508643 450 318.10 CH ( 0.88667 0.11333 )  6) LoyalCH < 0.764572 172 188.90 CH ( 0.76163 0.23837 )  12) ListPriceDiff < 0.235 70 95.61 CH ( 0.57143 0.42857 ) \*  13) ListPriceDiff > 0.235 102 69.76 CH ( 0.89216 0.10784 ) \*  7) LoyalCH > 0.764572 278 86.14 CH ( 0.96403 0.03597 ) \* |

* 1. **Create a plot of the tree, and interpret the results.**

We may see that the most important indicator of “Purchase” appears to be “LoyalCH”, since the first branch differentiates the intensity of customer brand loyalty to CH. In fact, the top three nodes contain “LoyalCH”.

* 1. **Predict the response on the test data and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?**

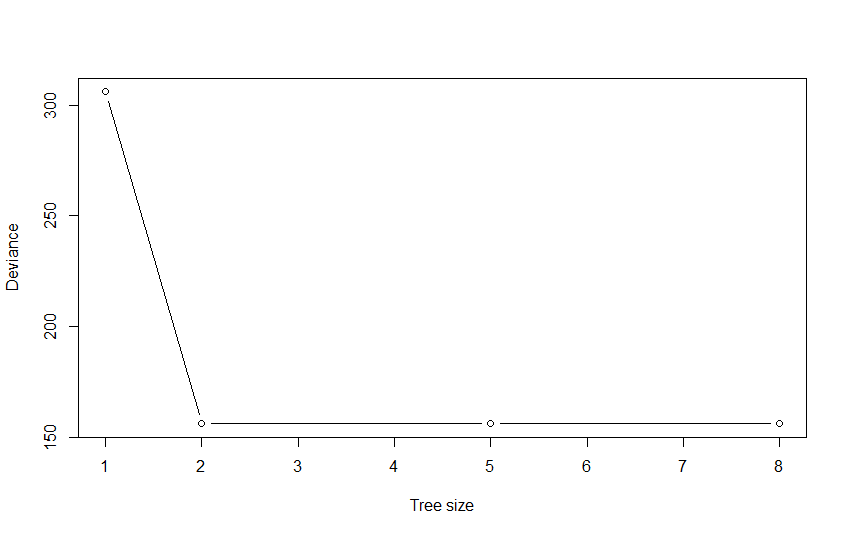
|  |  |  |
| --- | --- | --- |
| **tree.pred** | **CH** | **MM** |
| **CH** | 147 | 49 |
| **MM** | 12 | 62 |

The test error rate is 0.2259.

* 1. **Apply the cv.tree() function to the training set in order to determine the optimal tree size.**

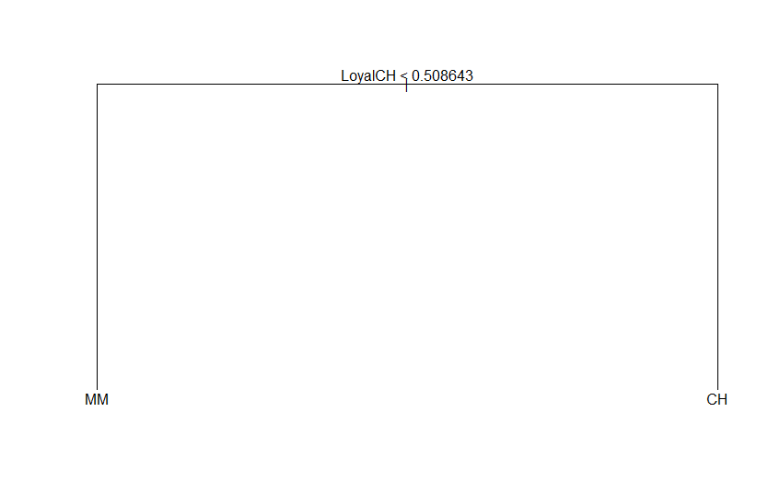
|  |
| --- |
| Results: |
| $size  [1] 8 5 2 1  $dev  [1] 156 156 156 306  $k  [1] -Inf 0.000000 4.666667 160.000000  $method  [1] "misclass"  attr(,"class")  [1] "prune" "tree.sequence" |

The optimal tree size is 2 or 5.

* 1. **Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.**
  2. **Which tree size corresponds to the lowest cross-validated classification error rate?**

We may see that the 2-node tree is the smallest tree with the lowest classification error rate.

* 1. **Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.**



* 1. **Compare the training error rates between the pruned and unpruned trees. Which is higher?**

|  |
| --- |
| Unpruned tree: |
| Classification tree:  tree(formula = Purchase ~ ., data = OJ.train)  Variables actually used in tree construction:  [1] "LoyalCH" "PriceDiff" "SpecialCH" "ListPriceDiff"  Number of terminal nodes: 8  Residual mean deviance: 0.7305 = 578.6 / 792  Misclassification error rate: 0.165 = 132 / 800 |

|  |
| --- |
| Pruned tree: |
| Classification tree:  snip.tree(tree = tree.oj, nodes = c(3L, 2L))  Variables actually used in tree construction:  [1] "LoyalCH"  Number of terminal nodes: 2  Residual mean deviance: 0.9115 = 727.4 / 798  Misclassification error rate: 0.1825 = 146 / 800 |

The test error of the tree is 0.165 and the test error of the pruned tree is 0.1825.

* 1. **Compare the test error rates between the pruned and unpruned trees. Which is higher?**

|  |  |  |
| --- | --- | --- |
| **prune.pred** | **CH** | **MM** |
| **CH** | 119 | 30 |
| **MM** | 40 | 81 |

In this case, the pruning process increased the test error rate to about 0.2593, but it produced a way more interpretable tree.

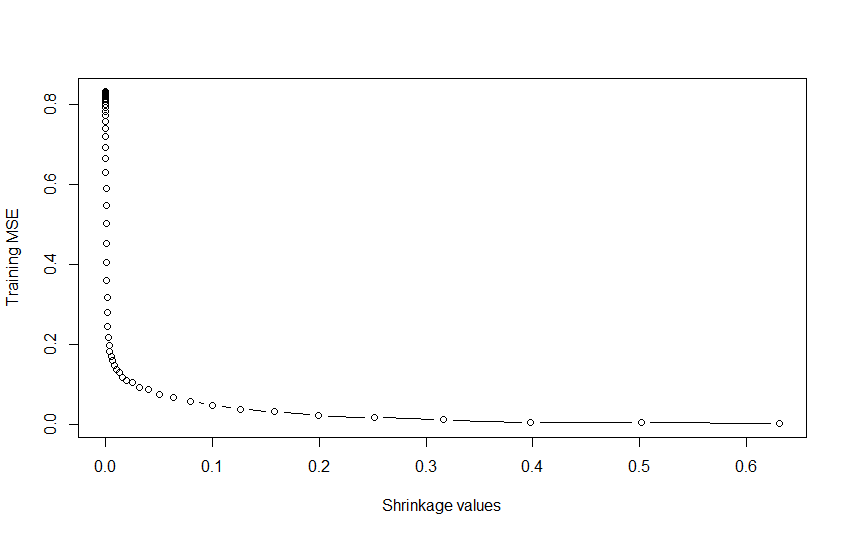
1. **We now use boosting to predict Salary in the Hitters data set.**
2. **Remove the observations for whom the salary information is unknown, and then log-transform the salaries.**

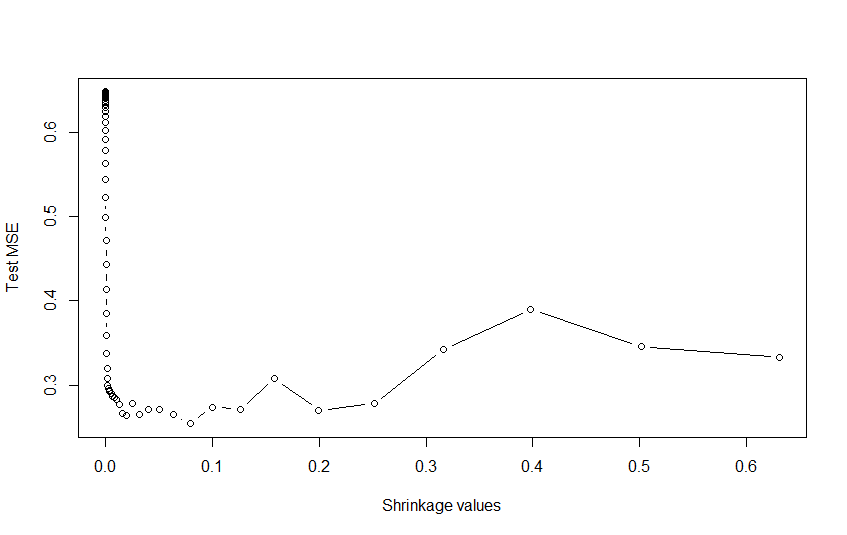
|  |
| --- |
| R codes: |
| >Hitters <- na.omit(Hitters);Hitters$Salary <- log(Hitters$Salary) |

1. **Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.**

|  |
| --- |
| R codes: |
| >train <- 1:200  >Hitters.train <- Hitters[train, ];Hitters.test <- Hitters[-train, ] |

1. **Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.**



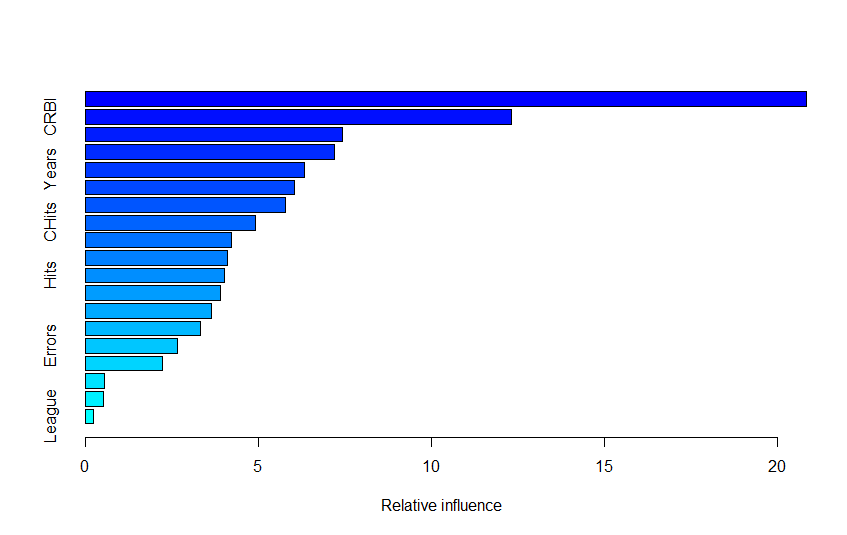
1. **Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.**

The lowest test error is 0.2540 when lambda is 0.0794.

1. **Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.**

The test MSE of boosting is 0.25 but the test MSE of the linear regression is 0.4918 and the test MSE of the ridge regression is 0.4570. As a result, the test MSE of boosting is lower than that of the linear regression and the ridge regression.

1. **Which variables appear to be the most important predictors in the boosted model?**



|  |
| --- |
| Results: |
| var rel.inf  CAtBat CAtBat 20.8404970 CRBI CRBI 12.3158959  Walks Walks 7.4186037 PutOuts PutOuts 7.1958539  Years Years 6.3104535 CWalks CWalks 6.0221656  CHmRun CHmRun 5.7759763 CHits CHits 4.8914360  AtBat AtBat 4.2187460 RBI RBI 4.0812410  Hits Hits 4.0117255 Assists Assists 3.8786634  HmRun HmRun 3.6386178 CRuns CRuns 3.3230296  Errors Errors 2.6369128 Runs Runs 2.2048386  Division Division 0.5347342 NewLeague NewLeague 0.4943540  League League 0.2062551 |

“CAtBat” is by far the most important variable.

1. **Now apply bagging to the training set. What is the test set MSE for this approach?**

The test set MSE is 0.2299.

river?

Discussion

We applied various tree-based approaches to many models. In the datasets we used, the bagging model was not the best in all cases. So, we can conclude that the best models fitting to the dataset can be different in many ways and we should not believe that the bagging or boosting model is always the best. In the Kaggle competition, the xgboost model is usually very effective for predicting. However, we should check the various models’ test errors and compare the models because there can be exceptions.

Appendix (R

**R codes**

**#ex 8,9,10**

**##8----------------------------------------------------------------------**

**###a)------------------------------------------------------**

**library(ISLR)**

**set.seed(1)**

**train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)**

**Carseats.train <- Carseats[train, ];Carseats.test <- Carseats[-train, ]**

**###b)------------------------------------------------------**

**library(tree)**

**tree.carseats <- tree(Sales ~ ., data = Carseats.train)**

**summary(tree.carseats)**

**plot(tree.carseats);text(tree.carseats, pretty = 0)**

**yhat <- predict(tree.carseats, newdata = Carseats.test)**

**mean((yhat - Carseats.test$Sales)^2)**

**###c)------------------------------------------------------**

**cv.carseats <- cv.tree(tree.carseats)**

**plot(cv.carseats$size, cv.carseats$dev, type = "b")**

**tree.min <- which.min(cv.carseats$dev)**

**points(tree.min, cv.carseats$dev[tree.min], col = "red", cex = 2, pch = 20)**

**prune.carseats <- prune.tree(tree.carseats, best = 9)**

**plot(prune.carseats);text(prune.carseats, pretty = 0)**

**yhat <- predict(prune.carseats, newdata = Carseats.test)**

**mean((yhat - Carseats.test$Sales)^2)**

**###d)------------------------------------------------------**

**bag.carseats <- randomForest(Sales ~ ., data = Carseats.train, mtry = 10, ntree = 500, importance = TRUE)**

**yhat.bag <- predict(bag.carseats, newdata = Carseats.test)**

**mean((yhat.bag - Carseats.test$Sales)^2)**

**importance(bag.carseats);varImpPlot(bag.carseats)**

**###e)------------------------------------------------------**

**rf.carseats <- randomForest(Sales ~ ., data = Carseats.train, mtry = 3, ntree = 500, importance = TRUE)**

**yhat.rf <- predict(rf.carseats, newdata = Carseats.test)**

**mean((yhat.rf - Carseats.test$Sales)^2)**

**importance(rf.carseats) ; varImpPlot(rf.carseats)**

**##9----------------------------------------------------------------------**

**###a)------------------------------------------------------**

**set.seed(1)**

**train <- sample(1:nrow(OJ), 800)**

**OJ.train <- OJ[train, ];OJ.test <- OJ[-train, ]**

**###b)------------------------------------------------------**

**tree.oj <- tree(Purchase ~ ., data = OJ.train);summary(tree.oj)**

**###c)------------------------------------------------------**

**tree.oj ;**

**###d)------------------------------------------------------**

**plot(tree.oj);text(tree.oj, pretty = 0)**

**###e)------------------------------------------------------**

**tree.pred <- predict(tree.oj, OJ.test, type = "class")**

**table(tree.pred, OJ.test$Purchase) ; 1 - (147 + 62) / 270**

**###f)------------------------------------------------------**

**cv.oj <- cv.tree(tree.oj, FUN = prune.misclass);cv.oj**

**###g)------------------------------------------------------**

**plot(cv.oj$size, cv.oj$dev, type = "b", xlab = "Tree size", ylab = "Deviance")**

**###i)------------------------------------------------------**

**prune.oj <- prune.misclass(tree.oj, best = 2)**

**plot(prune.oj);text(prune.oj, pretty = 0)**

**###j)------------------------------------------------------**

**summary(tree.oj);summary(prune.oj)**

**###k)------------------------------------------------------**

**prune.pred <- predict(prune.oj, OJ.test, type = "class")**

**table(prune.pred, OJ.test$Purchase)**

**1 - (119 + 81) / 270**

**##10----------------------------------------------------------------------**

**###a)------------------------------------------------------**

**Hitters <- na.omit(Hitters);Hitters$Salary <- log(Hitters$Salary)**

**###b)------------------------------------------------------**

**train <- 1:200**

**Hitters.train <- Hitters[train, ];Hitters.test <- Hitters[-train, ]**

**###c)------------------------------------------------------**

**library(gbm);set.seed(1)**

**pows <- seq(-10, -0.2, by = 0.1);lambdas <- 10^pows**

**train.err <- rep(NA, length(lambdas))**

**for (i in 1:length(lambdas)) {**

**boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[i])**

**pred.train <- predict(boost.hitters, Hitters.train, n.trees = 1000)**

**train.err[i] <- mean((pred.train - Hitters.train$Salary)^2)**

**}**

**plot(lambdas, train.err, type = "b", xlab = "Shrinkage values", ylab = "Training MSE")**

**###d)------------------------------------------------------**

**set.seed(1)**

**test.err <- rep(NA, length(lambdas))**

**for (i in 1:length(lambdas)) {**

**boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[i])**

**yhat <- predict(boost.hitters, Hitters.test, n.trees = 1000)**

**test.err[i] <- mean((yhat - Hitters.test$Salary)^2)**

**}**

**plot(lambdas, test.err, type = "b", xlab = "Shrinkage values", ylab = "Test MSE")**

**min(test.err);lambdas[which.min(test.err)]**

**###e)------------------------------------------------------**

**library(glmnet)**

**fit1 <- lm(Salary ~ ., data = Hitters.train);pred1 <- predict(fit1, Hitters.test)**

**mean((pred1 - Hitters.test$Salary)^2)**

**x <- model.matrix(Salary ~ ., data = Hitters.train)**

**x.test <- model.matrix(Salary ~ ., data = Hitters.test)**

**y <- Hitters.train$Salary**

**fit2 <- glmnet(x, y, alpha = 0);pred2 <- predict(fit2, s = 0.01, newx = x.test)**

**mean((pred2 - Hitters.test$Salary)^2)**

**###f)------------------------------------------------------**

**library(gbm)**

**boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[which.min(test.err)]);summary(boost.hitters)**

**###g)------------------------------------------------------**

**set.seed(1)**

**bag.hitters <- randomForest(Salary ~ ., data = Hitters.train, mtry = 19, ntree = 500)**

**yhat.bag <- predict(bag.hitters, newdata = Hitters.test)**

**mean((yhat.bag - Hitters.test$Salary)^2)**