Non-linear Models for Classification

04/30/2021

Seungheon Han

1. Use KNN & SVM to predict whether a given car gets high or low gas mileage based on the `Auto` data set.

(a) Create a binary variable `mpg01`, and generate the training and test sets

```
pacman::p_load(ISLR, GGally, ggplot2, dplyr, class, e1071,caret,tree, gbm, randomForest)


auto <- Auto[, setdiff(names(Auto), c("name"))]
auto$origin <- as.factor(auto$origin)
auto$cylinders <- as.factor(auto$cylinders)
str(auto)
```

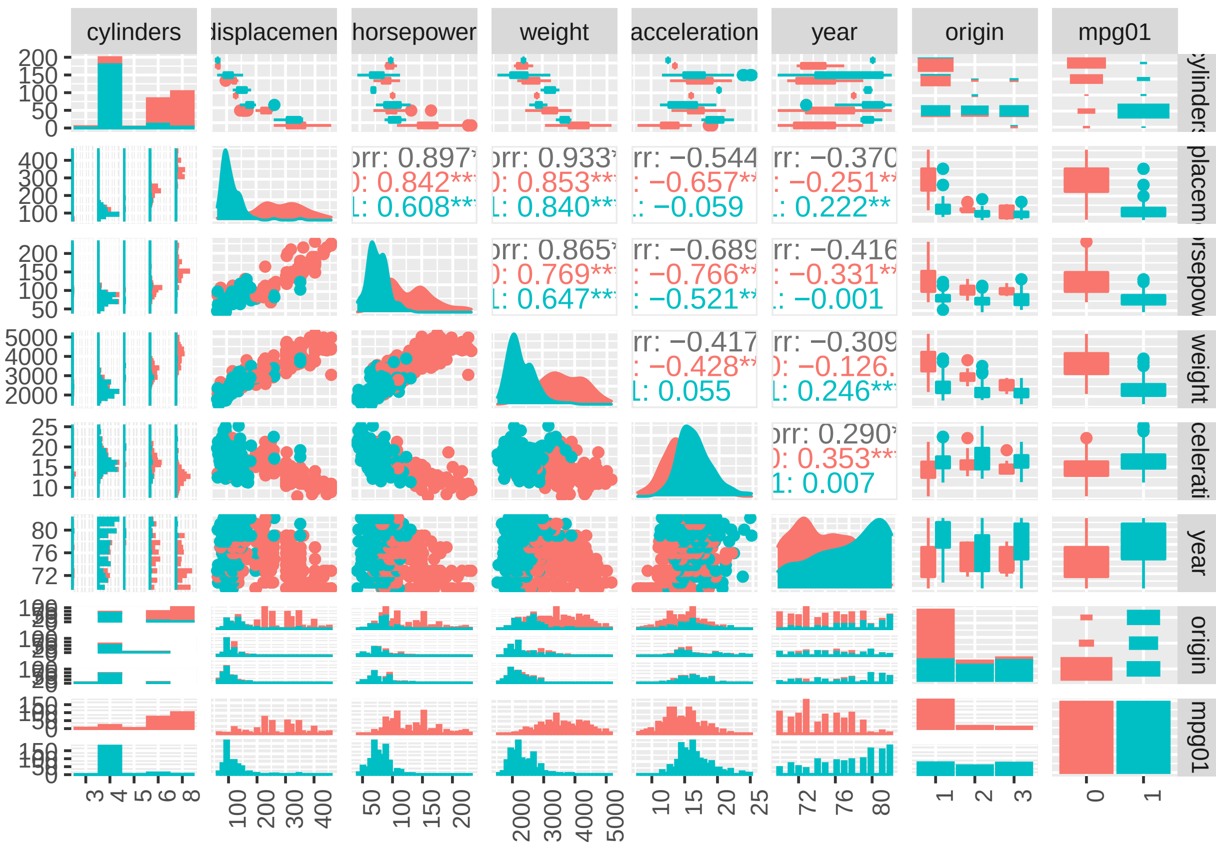
```
## 'data.frame':    392 obs. of  8 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : Factor w/ 5 levels "3","4","5","6",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
```

```
auto$mpg01 <- ifelse(auto$mpg > median(auto$mpg), 1,0)
auto <- auto[, setdiff(names(auto), c("mpg"))]
auto$mpg01 <- as.factor(auto$mpg01)


set.seed(123)
split <- sample(2, nrow(auto), replace=T, prob = c(0.8, 0.2))
train <- auto[split == 1,]
test <- auto[split == 2,]
```

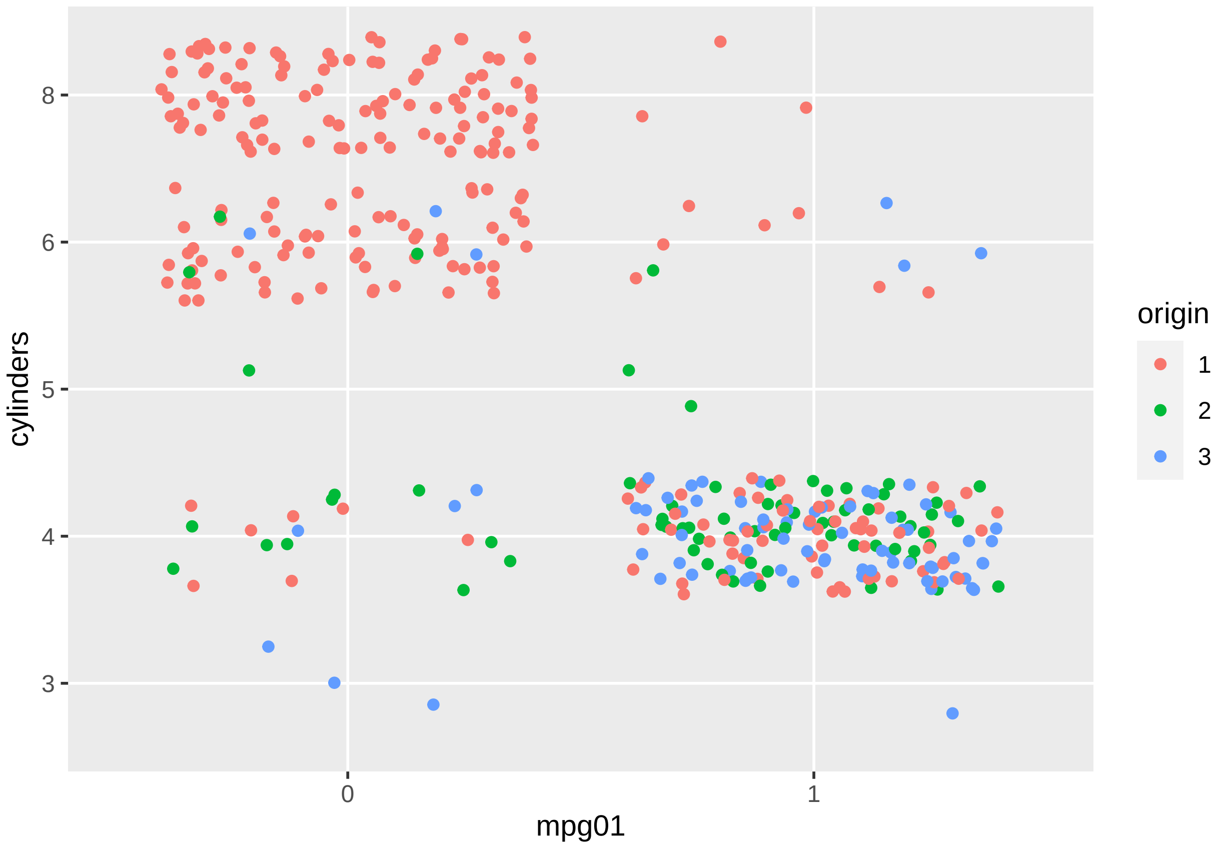
(b) Perform KNN on the training data, with several values of K, in order to predict `mpg01`. Use only the variables that seemed most associated with `mpg01`.

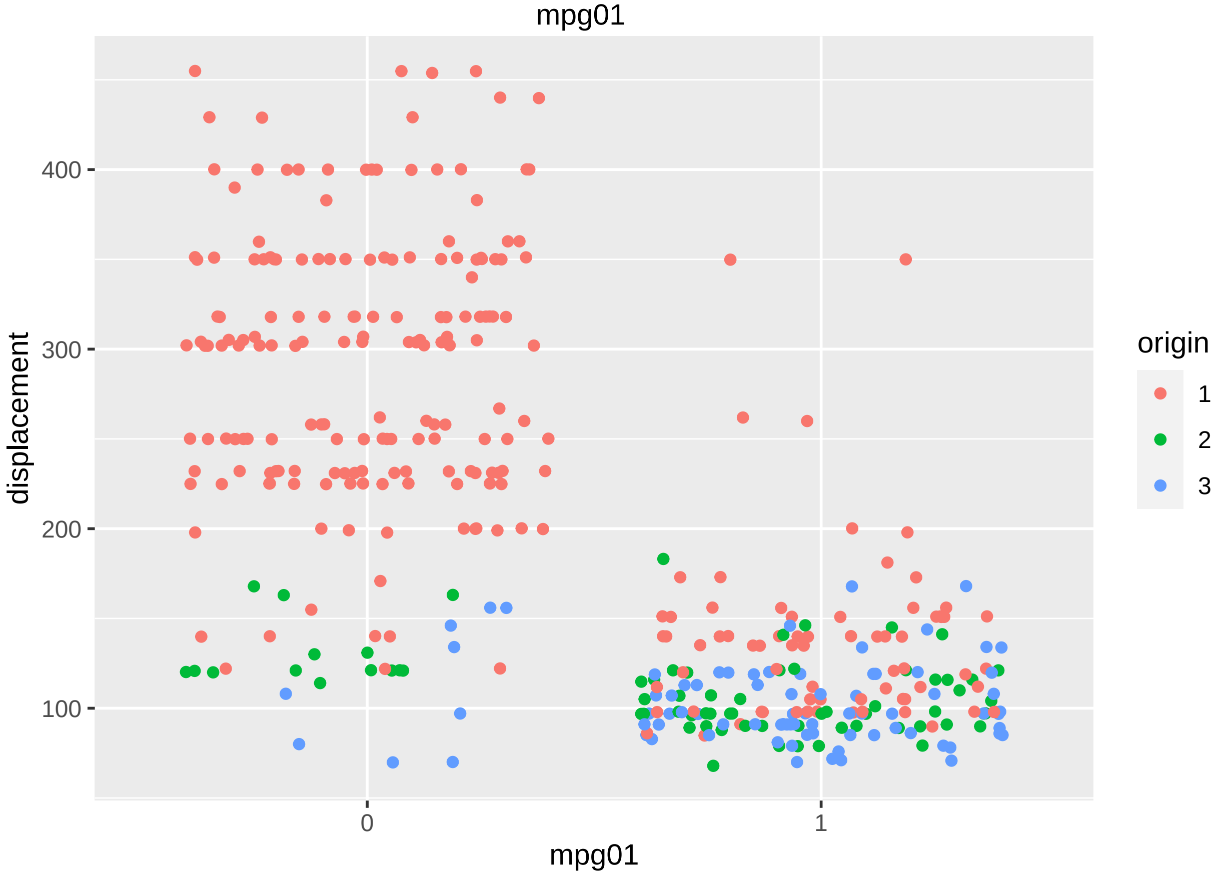
```
### exploring the relationships between mpg01 and each of the predictor
auto %>%
  ggpairs(aes(col = mpg01, fill = mpg01)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

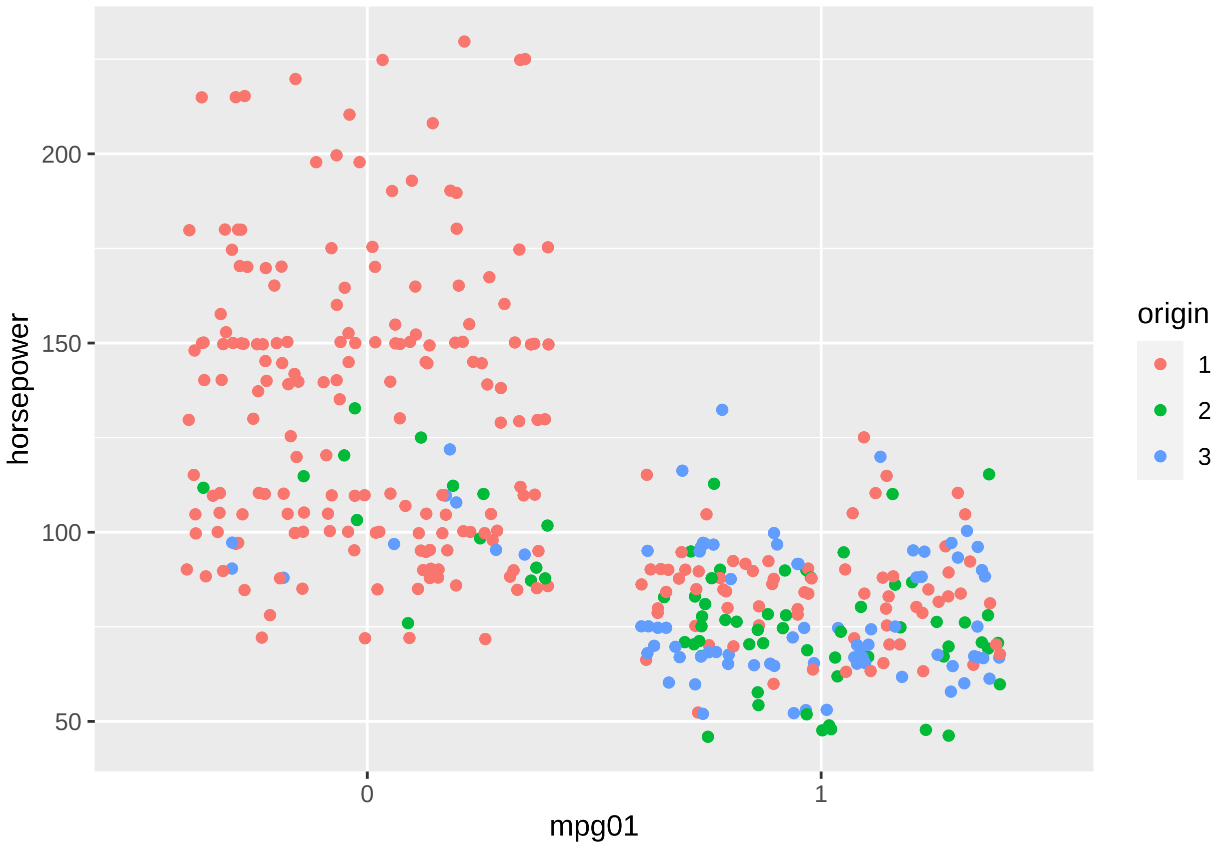
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

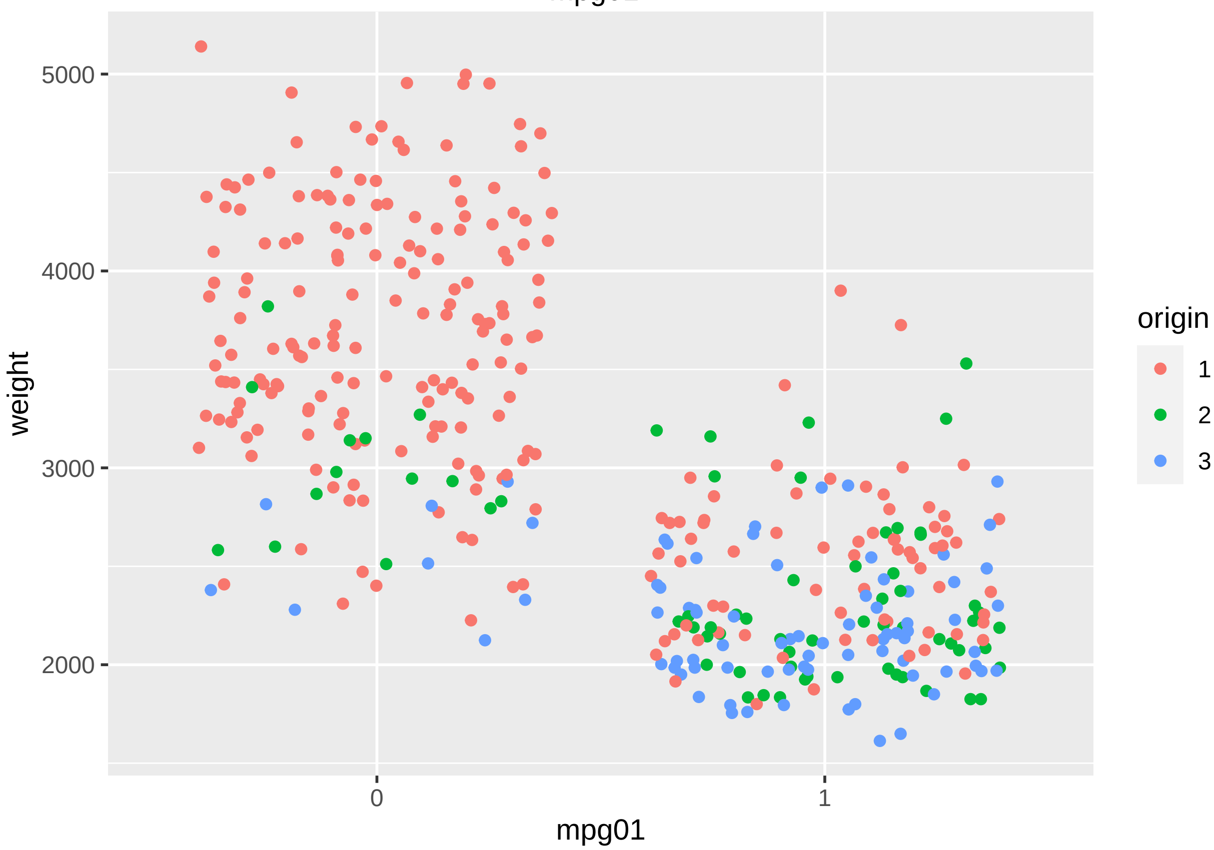


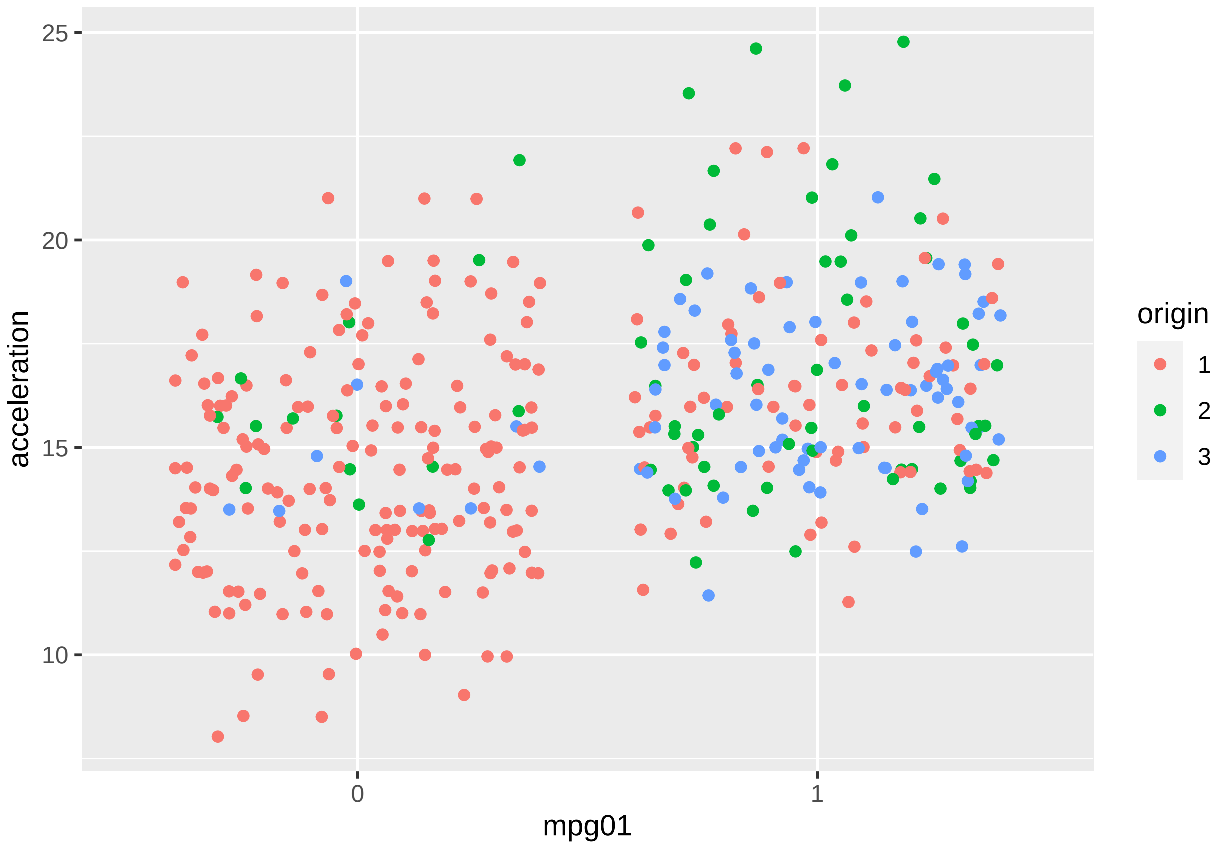

```r
for(i in 1:ncol(auto)){
  print(ggplot(auto, aes_string("mpg01", colnames(auto)[i]))+
          geom_jitter(aes(col=origin)))
}
```

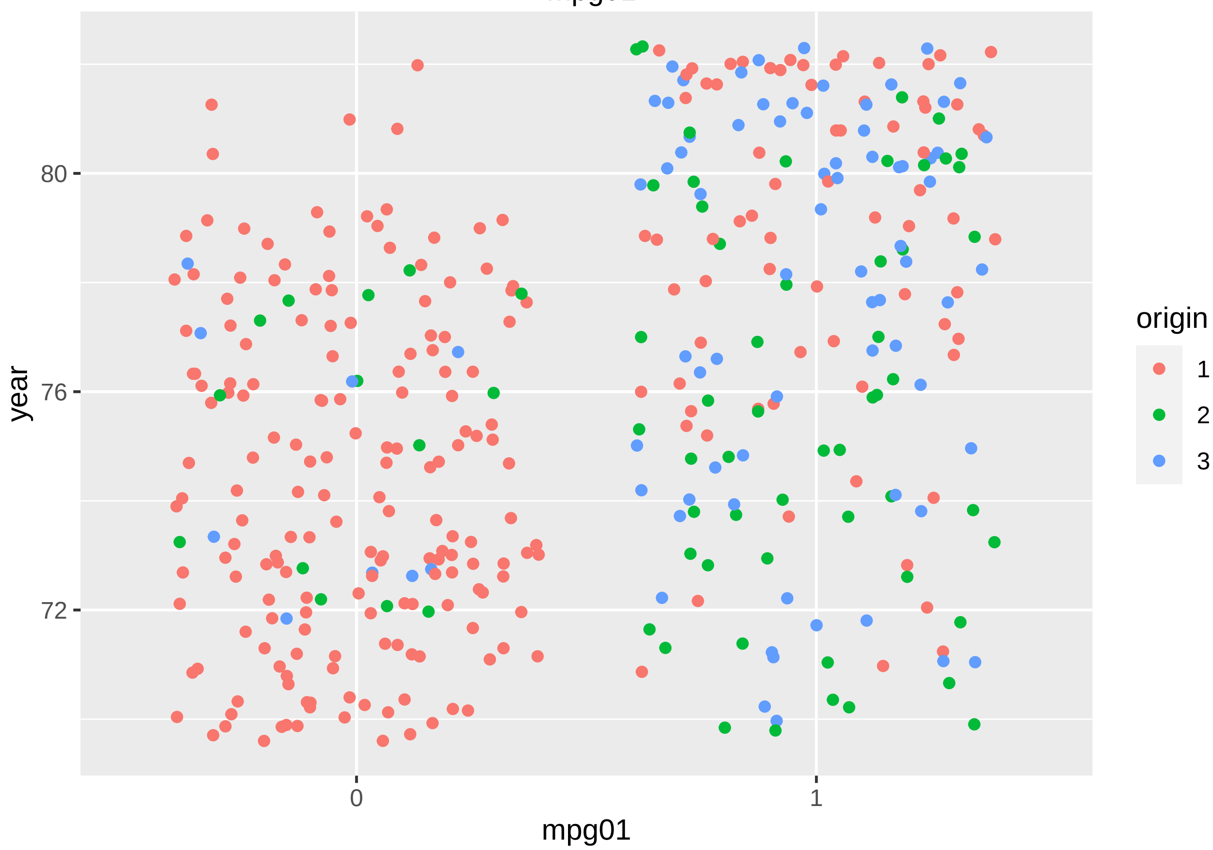

cylinders
8
6
5
4
3
0
1
mpg01
origin
1
2
3

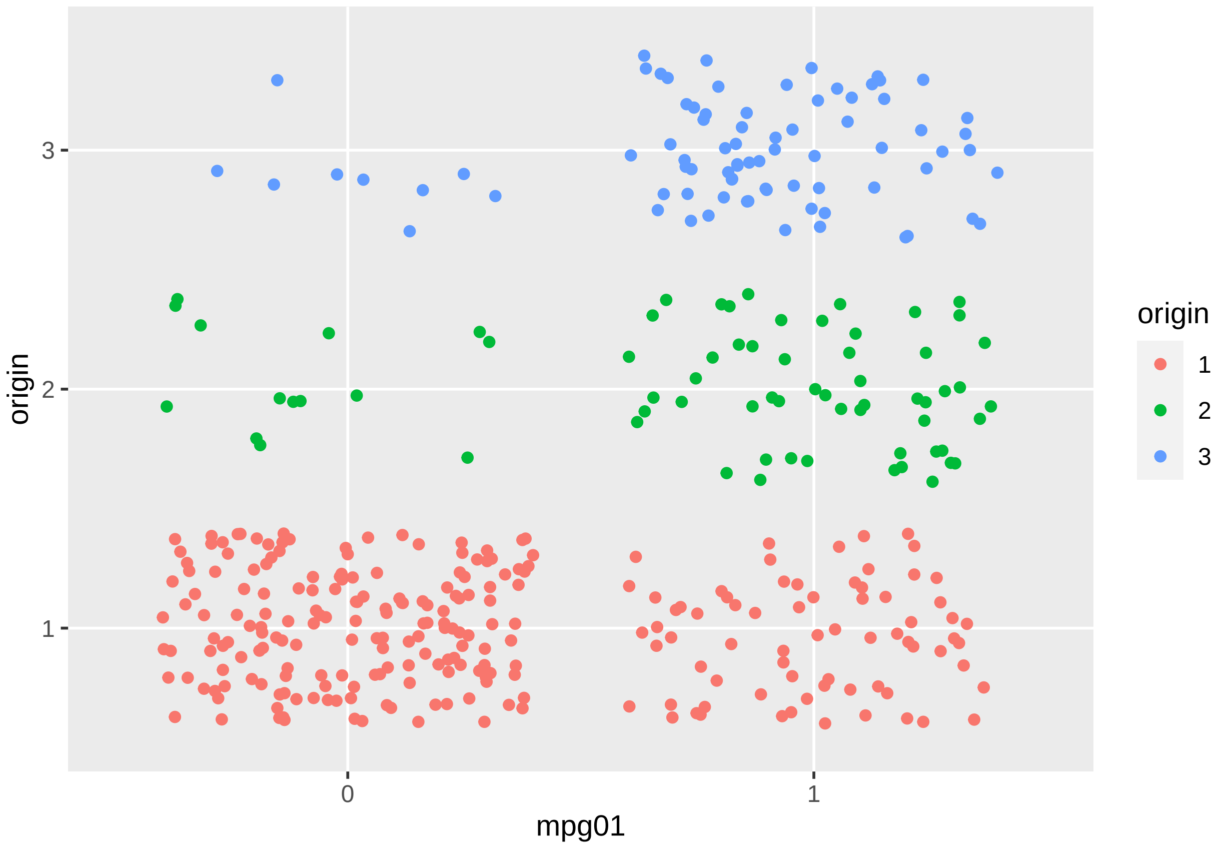

displacement
400
300
200
100
0
1
mpg01
origin
1
2
3


horsepower
200
150
100
50
0
1
mpg01
origin
1
2
3

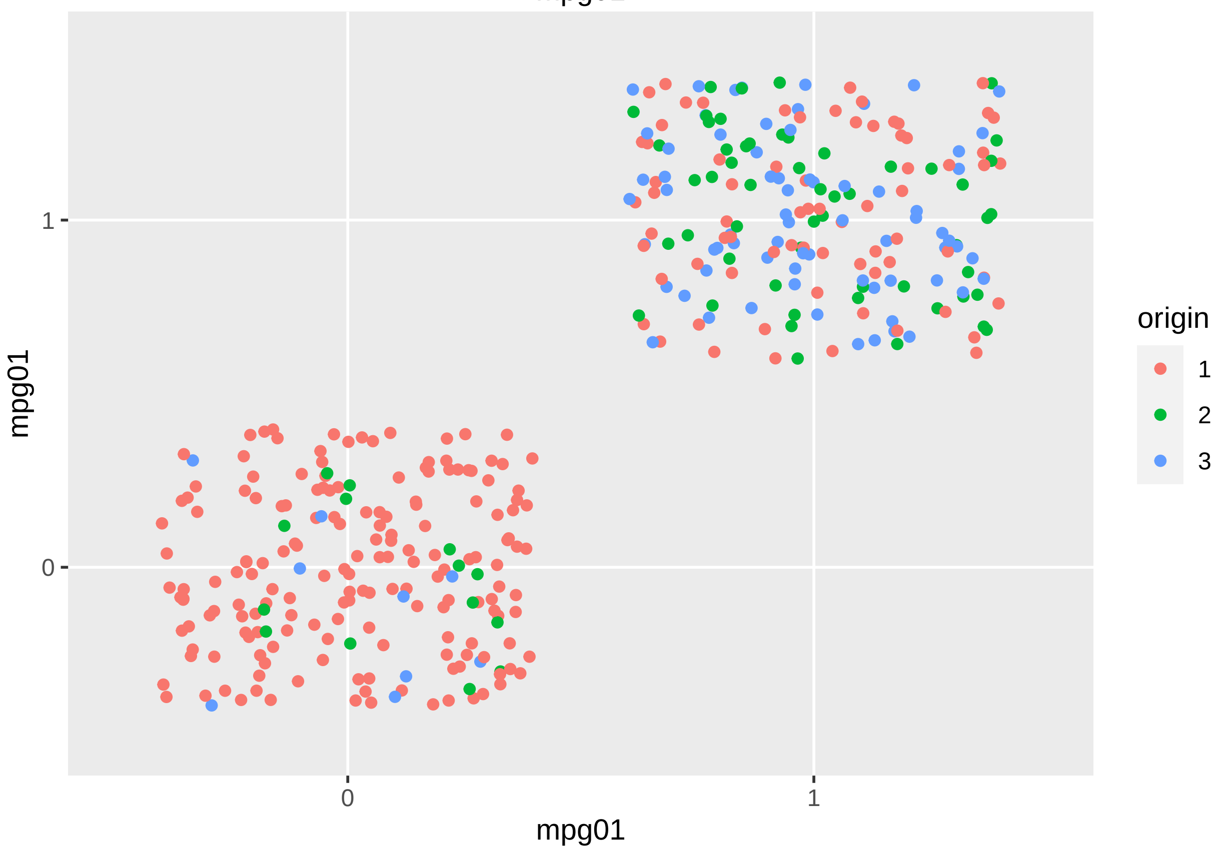

weight
5000
4000
3000
2000
0
1
mpg01
origin
1
2
3


acceleration
mpg01
origin
1
2
3
25
20
15
10
0
1


year
mpg01
origin
1
2
3
80
76
72
0
1



origin
3
2
1
0
1
mpg01
origin
1
2
3



mpg01
1
0
0
1
mpg01
origin
1
2
3

```r
### the most associated predictors: cylinders, displacement, horsepower, weight, year
train <- train %>%
  select(-c("acceleration", "origin"))
test <- test %>%
  select(-c("acceleration", "origin"))

train_x <- as.data.frame(train[,setdiff(names(train), c("mpg01"))])
train_y <- train[, "mpg01"]
test_x <- as.data.frame(test[, setdiff(names(test), c("mpg01"))])
test_y <- test[, "mpg01"]
### k-fold CV for find the best K in KNN model
set.seed(123)
K.knn = c(1:20)
k.fold <- 10
folds <- createFolds(train_y, k = k.fold)

error.vec = NULL
for(i in K.knn){
  error.list = NULL
  for(j in 1:k.fold){
    knn.pred <- knn(train_x[-unlist(folds[j]),], train_x[unlist(folds[j]),], train_y[-unlist(folds[j])]
    cv.error <- sum(knn.pred != train_y[unlist(folds[j])])/length(unlist(folds[j]))
    error.list <- c(error.list, cv.error)
  }
  avg.error <- round(mean(error.list),4)
  error.vec <- c(error.vec, avg.error)
  print(paste("K =", i, "cv-error:", avg.error))
}
```

```
## [1] "K = 1 cv-error: 0.1226"
## [1] "K = 2 cv-error: 0.1354"
## [1] "K = 3 cv-error: 0.1356"
## [1] "K = 4 cv-error: 0.1261"
## [1] "K = 5 cv-error: 0.1163"
## [1] "K = 6 cv-error: 0.1226"
## [1] "K = 7 cv-error: 0.1226"
## [1] "K = 8 cv-error: 0.1162"
## [1] "K = 9 cv-error: 0.1228"
## [1] "K = 10 cv-error: 0.126"
## [1] "K = 11 cv-error: 0.1259"
## [1] "K = 12 cv-error: 0.1386"
## [1] "K = 13 cv-error: 0.1355"
## [1] "K = 14 cv-error: 0.1355"
## [1] "K = 15 cv-error: 0.1386"
## [1] "K = 16 cv-error: 0.1385"
## [1] "K = 17 cv-error: 0.1449"
## [1] "K = 18 cv-error: 0.1353"
## [1] "K = 19 cv-error: 0.1353"
## [1] "K = 20 cv-error: 0.1385"
```

```r
print(paste("The K minimizing the CV error: K =", match(min(error.vec), error.vec), "with error", min(e
```

```
## [1] "The K minimizing the CV error: K = 8 with error 0.1162"
```

```r
# tune.knn function generates the same result
set.seed(123)
tune.out.knn <- tune.knn(x = train_x, y = train_y, k=1:20)
print(paste("Best K:", tune.out.knn$best.parameters))
```

```
## [1] "Best K: 8"
```

```r
### comment ###
# K=8 generates the smallest validation error.
```

(c) Fit a support vector classifier to the training data with various values of `cost`, in order to predict whether a car gets high or low gas mileage.

```r
### (c) ###
set.seed(123)
tune.out.lin <- tune(svm, mpg01~., data = train, kernal = "linear",
                     ranges = list(cost = c(0.1, 1, 10, 100, 1000, 10000,30000)))

### CV-errors by different costs
summary(tune.out.lin)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  10000
##
## - best performance: 0.08165323
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-01 0.10352823 0.05078718
## 2 1e+00 0.10665323 0.05516446
## 3 1e+01 0.08800403 0.04363379
## 4 1e+02 0.08487903 0.02960729
## 5 1e+03 0.08165323 0.02604546
## 6 1e+04 0.08165323 0.03015969
## 7 3e+04 0.09465726 0.05679151
```

```r
### best cost
bestmod <- tune.out.lin$best.model
print(paste("the best cost is", summary(bestmod)$cost))
```

```
## [1] "the best cost is 10000"
```

```r
### fit the svm model
svm.fit.lin <- svm(formula= mpg01~., data = train, kernal = "linear",
                   cost = 10000)
summary(svm.fit.lin)
```

```
##
## Call:
## svm(formula = mpg01 ~ ., data = train, kernal = "linear", cost = 10000)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10000
##
## Number of Support Vectors:  51
##
##  ( 29 22 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Comments:
with support vector classifier, the best cost is 10000

(d) Now repeat (c), this time using SVMs with radial and polynomial basis kernels, with different values of `gamma` and `degree` and cost.

```
### kernal = radial
set.seed(123)
tune.out.rad <- tune(svm, mpg01~., data=train, kernel="radial",
              ranges=list(cost=c(0.1,1,10,100,1000),
                          gamma=c(0.5,1,2,3,4) ))

summary(tune.out.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     3
##
## - best performance: 0.06270161
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.10040323 0.07010733
## 2  1e+00   0.5 0.09102823 0.04960171
## 3  1e+01   0.5 0.07852823 0.02179456
## 4  1e+02   0.5 0.07237903 0.02984280
## 5  1e+03   0.5 0.07570565 0.04797571
## 6  1e-01   1.0 0.09415323 0.05285737
```

```
## 7  1e+00   1.0 0.07842742 0.04461323
## 8  1e+01   1.0 0.06905242 0.03532623
## 9  1e+02   1.0 0.06633065 0.04146783
## 10 1e+03   1.0 0.08820565 0.04911492
## 11 1e-01   2.0 0.09727823 0.04730450
## 12 1e+00   2.0 0.06592742 0.03729748
## 13 1e+01   2.0 0.07883065 0.03455569
## 14 1e+02   2.0 0.08205645 0.04071882
## 15 1e+03   2.0 0.09758065 0.04587598
## 16 1e-01   3.0 0.10362903 0.04645077
## 17 1e+00   3.0 0.06270161 0.03879790
## 18 1e+01   3.0 0.06945565 0.05156250
## 19 1e+02   3.0 0.07883065 0.05236073
## 20 1e+03   3.0 0.09133065 0.05048202
## 21 1e-01   4.0 0.13235887 0.05948065
## 22 1e+00   4.0 0.06602823 0.04996957
## 23 1e+01   4.0 0.06945565 0.05561222
## 24 1e+02   4.0 0.08195565 0.05424859
## 25 1e+03   4.0 0.08820565 0.04685362
```

```r
### best gamma & cost
bestmod2 <- tune.out.rad$best.parameters
print(paste("best cost:", bestmod2$cost,"& best gamma:", bestmod2$gamma))
```

```
## [1] "best cost: 1 & best gamma: 3"
```

```r
### kernal = polynomial
set.seed(123)
tune.out.poly <- tune(svm, mpg01~., data = train, kernal = "polynomial",
                      ranges = list(cost = c(0.1,1,10,100,1000),
                                    degree = c(1:10)))
summary(tune.out.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      1
##
## - best performance: 0.08165323
##
## - Detailed performance results:
##     cost degree      error dispersion
## 1  1e-01      1 0.10352823 0.05078718
## 2  1e+00      1 0.10665323 0.05516446
## 3  1e+01      1 0.08800403 0.04363379
## 4  1e+02      1 0.08487903 0.02960729
## 5  1e+03      1 0.08165323 0.02604546
## 6  1e-01      2 0.10352823 0.05078718
## 7  1e+00      2 0.10665323 0.05516446
```

```
## 8  1e+01      2 0.08800403 0.04363379
## 9  1e+02      2 0.08487903 0.02960729
## 10 1e+03      2 0.08165323 0.02604546
## 11 1e-01      3 0.10352823 0.05078718
## 12 1e+00      3 0.10665323 0.05516446
## 13 1e+01      3 0.08800403 0.04363379
## 14 1e+02      3 0.08487903 0.02960729
## 15 1e+03      3 0.08165323 0.02604546
## 16 1e-01      4 0.10352823 0.05078718
## 17 1e+00      4 0.10665323 0.05516446
## 18 1e+01      4 0.08800403 0.04363379
## 19 1e+02      4 0.08487903 0.02960729
## 20 1e+03      4 0.08165323 0.02604546
## 21 1e-01      5 0.10352823 0.05078718
## 22 1e+00      5 0.10665323 0.05516446
## 23 1e+01      5 0.08800403 0.04363379
## 24 1e+02      5 0.08487903 0.02960729
## 25 1e+03      5 0.08165323 0.02604546
## 26 1e-01      6 0.10352823 0.05078718
## 27 1e+00      6 0.10665323 0.05516446
## 28 1e+01      6 0.08800403 0.04363379
## 29 1e+02      6 0.08487903 0.02960729
## 30 1e+03      6 0.08165323 0.02604546
## 31 1e-01      7 0.10352823 0.05078718
## 32 1e+00      7 0.10665323 0.05516446
## 33 1e+01      7 0.08800403 0.04363379
## 34 1e+02      7 0.08487903 0.02960729
## 35 1e+03      7 0.08165323 0.02604546
## 36 1e-01      8 0.10352823 0.05078718
## 37 1e+00      8 0.10665323 0.05516446
## 38 1e+01      8 0.08800403 0.04363379
## 39 1e+02      8 0.08487903 0.02960729
## 40 1e+03      8 0.08165323 0.02604546
## 41 1e-01      9 0.10352823 0.05078718
## 42 1e+00      9 0.10665323 0.05516446
## 43 1e+01      9 0.08800403 0.04363379
## 44 1e+02      9 0.08487903 0.02960729
## 45 1e+03      9 0.08165323 0.02604546
## 46 1e-01     10 0.10352823 0.05078718
## 47 1e+00     10 0.10665323 0.05516446
## 48 1e+01     10 0.08800403 0.04363379
## 49 1e+02     10 0.08487903 0.02960729
## 50 1e+03     10 0.08165323 0.02604546
```

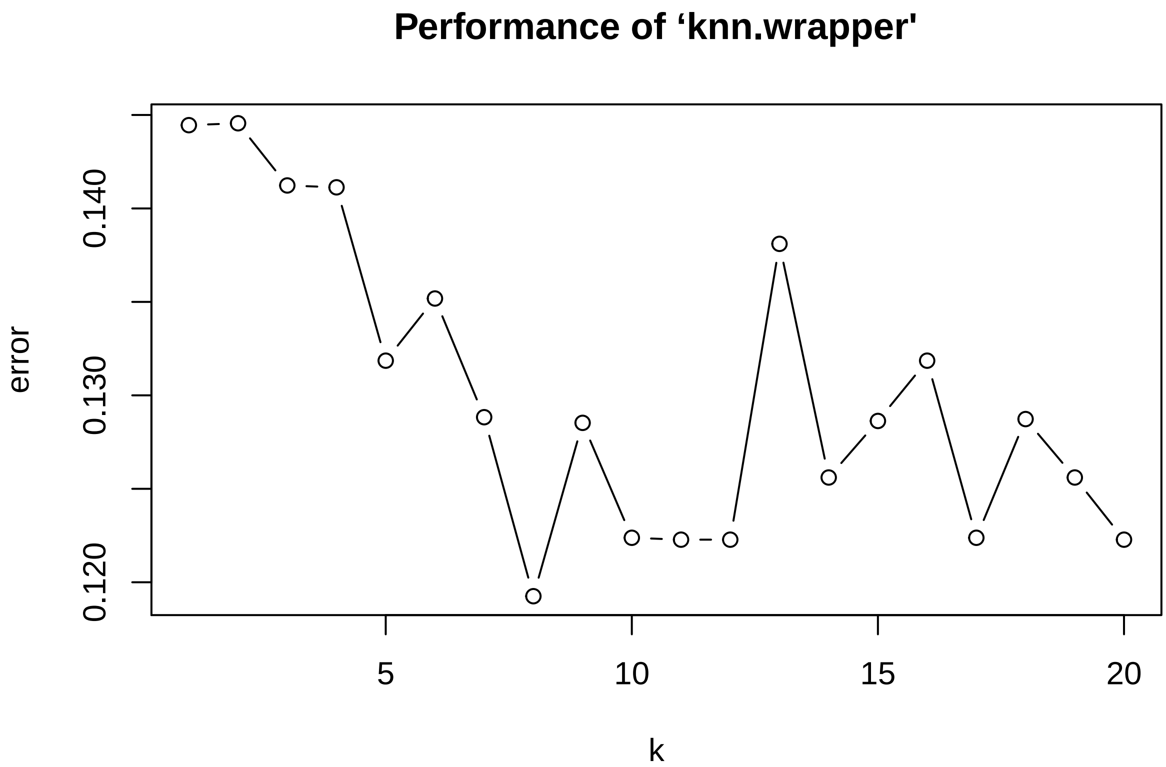
```r
bestmod3 <- tune.out.poly$best.parameters
print(paste("best cost:", bestmod3$cost,"& best degree:", bestmod3$degree))
```

```
## [1] "best cost: 1000 & best degree: 1"
```

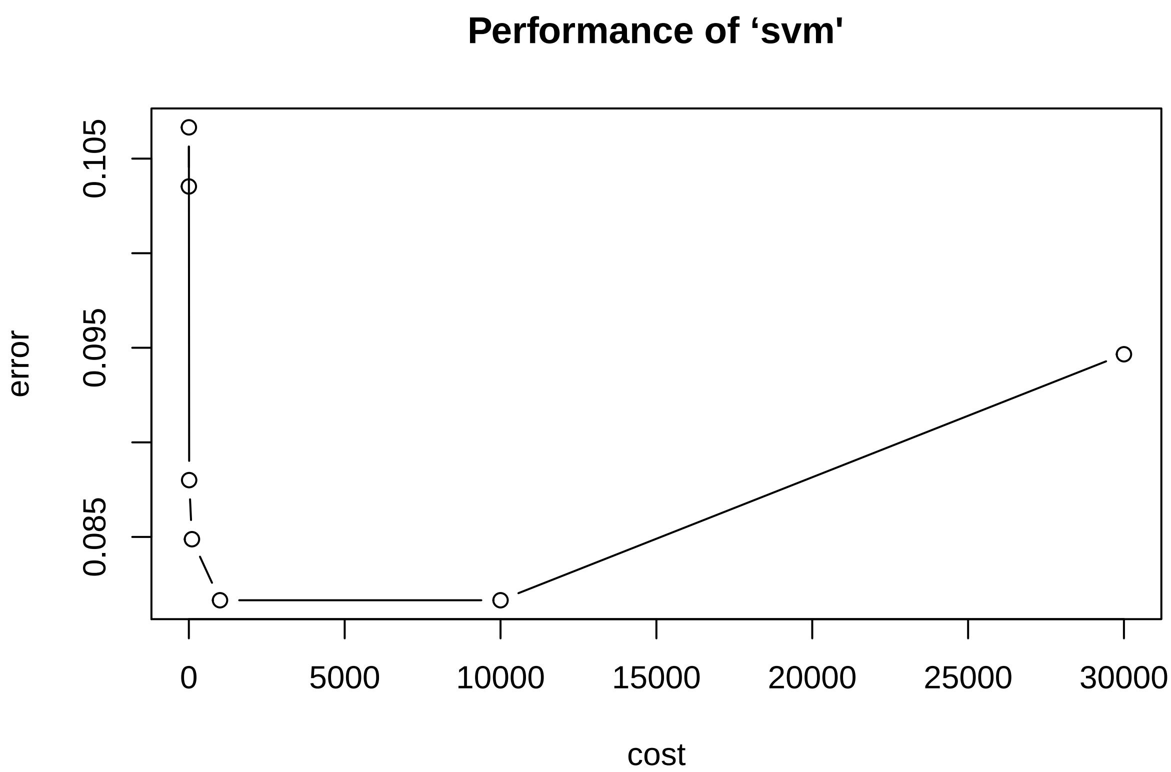
```
Comments:
for SVM with radial kernel, 10-fold CV finds the optimal cost and gamma
which have turned out to be 1 and 3 respectively. for SVM with polynomical kernel,
the best cost and degree are 1000 and 1 respectively.
```

(e) Make some plots to back up your assertions in (b), (c) and (d).

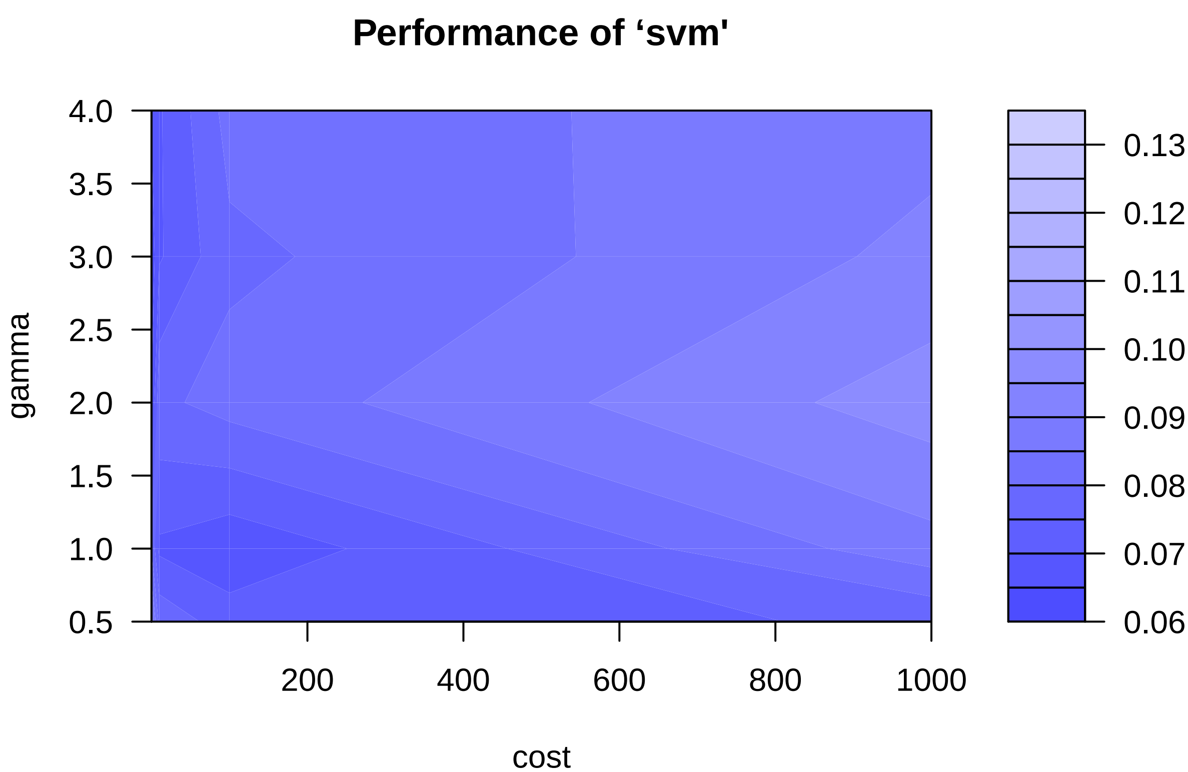
```r
### best K minimizing the validation error is 8
plot(tune.out.knn)
```



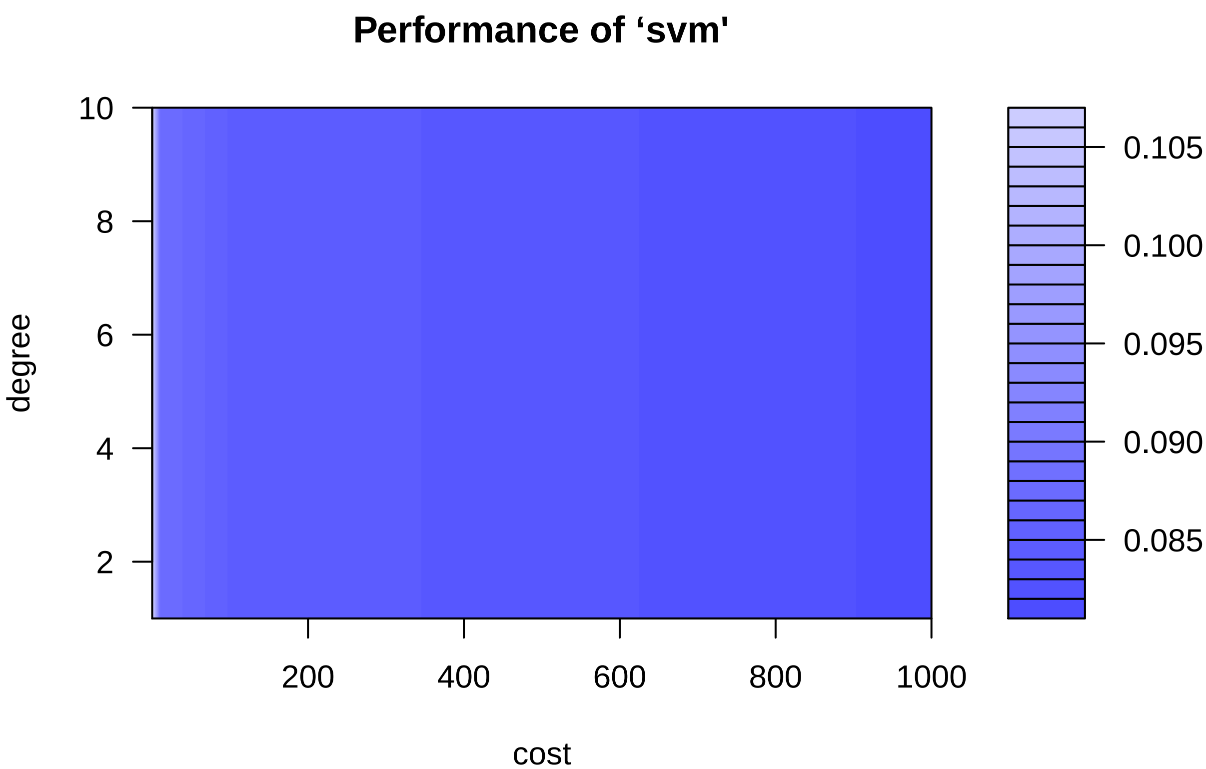

```r
### best cost minimizing the validation error is 10000
plot(tune.out.lin)
```




```r
### best cost is 100 and gamma is 3
plot(tune.out.rad)
```



```r
### best cost is 10000 and degree is 1
plot(tune.out.poly)
```



(f) Compare the test errors of the best tuned models for KNN, linear SVM, SVM with radial basis kernel, and SVM with polynomial basis kernel.

```r
### (f) ###

### KNN model with K=8
optimal.knn <- knn(train_x, test_x, train_y, k=8)
table.knn <- table(optimal.knn, test_y)
error.knn <- 1-sum(diag(table.knn))/sum(table.knn)

### Support Vector Classifier with cost = 10000
svm.fit.lin <- svm(formula= mpg01~., data = train, kernal = "linear",
                   cost = 10000)
pred.lin.svm <- predict(svm.fit.lin, test_x)
table.lin.svm <- table(pred.lin.svm, test_y)
error.lin.svm <- 1-sum(diag(table.lin.svm))/sum(table.lin.svm)

### SVM with radial kernels with cost = 100, gamma = 3
svm.fit.rad <- svm(formula= mpg01~., data = train, kernal = "radial",
                   cost = 100, gamma = 3)
pred.rad.svm <- predict(svm.fit.rad, test_x)
table.rad.svm <- table(pred.rad.svm, test_y)
error.rad.svm <- 1-sum(diag(table.rad.svm))/sum(table.rad.svm)

### SVM with polynomial kernels with cost = 10000, degree = 1
```

```r
svm.fit.poly <- svm(formula= mpg01~., data = train, kernal = "polynomial",
                    cost = 1000, degree = 1)
pred.poly.svm <- predict(svm.fit.poly, test_x)
table.poly.svm <- table(pred.poly.svm, test_y)
error.poly.svm <- 1-sum(diag(table.poly.svm))/sum(table.poly.svm)

### Comparing the test errors generated from the four models
print(paste("test error of knn model:", round(error.knn,4)))
```

```
## [1] "test error of knn model: 0.1486"
```

```r
print(paste("test error of linear svm model:", round(error.lin.svm,4)))
```

```
## [1] "test error of linear svm model: 0.0946"
```

```r
print(paste("test error of radial svm model:", round(error.rad.svm,4)))
```

```
## [1] "test error of radial svm model: 0.1081"
```

```r
print(paste("test error of polynomial svm model:", round(error.poly.svm,4)))
```

```
## [1] "test error of polynomial svm model: 0.0811"
```

```r
### comment:
# the SVM model with polynomial basis kernel produces the smallest test error.
```

2. (Regression Tree, Boosting, Bagging and Random Forest) Use the `OJ` data set which is part of the `ISLR` package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```r
oj <- OJ
set.seed(1)
split.train <- sample(1:nrow(oj), 800)
train <- oj[split.train,]
test <- oj[-split.train,]
```

(b) Fit a tree to the training data, with `Purchase` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics about the tree, and describe the results obtained.

```r
tree.oj.train <- tree(Purchase ~., train)
summary(tree.oj.train)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
```

```
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## [5] "PctDiscMM"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7432 = 587.8 / 791
## Misclassification error rate: 0.1588 = 127 / 800
```

```
Comments:
terminal nodes = 9, error rate = 15.875%
```

(c) Pick one of the terminal nodes, and interpret the information displayed.

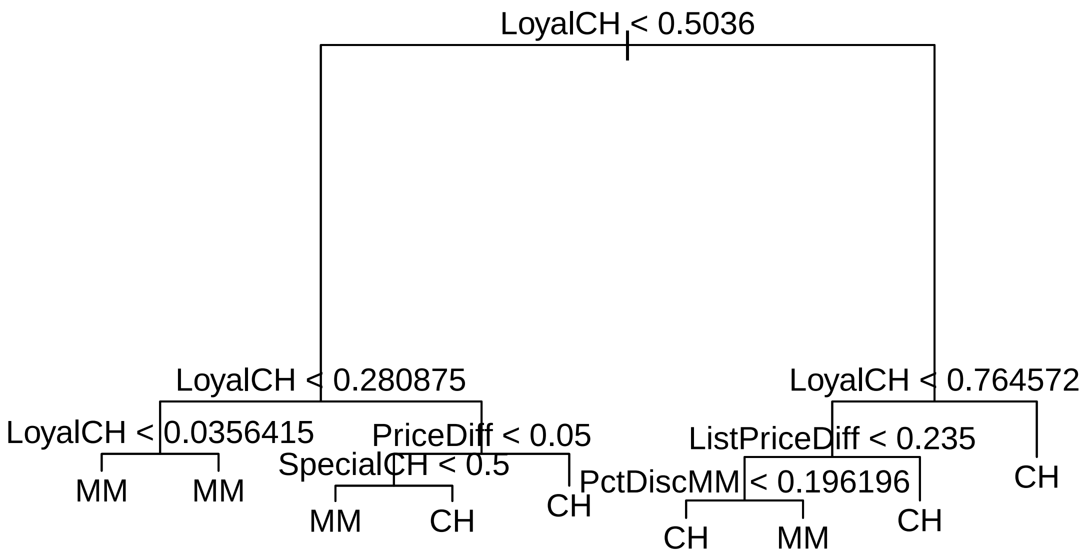
```r
tree.oj.train
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60625 0.39375 )
##    2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
##      4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
##        8) LoyalCH < 0.0356415 59   10.14 MM ( 0.01695 0.98305 ) *
##        9) LoyalCH > 0.0356415 118  116.40 MM ( 0.19492 0.80508 ) *
##      5) LoyalCH > 0.280875 188  258.00 MM ( 0.44149 0.55851 )
##       10) PriceDiff < 0.05 79   84.79 MM ( 0.22785 0.77215 )
##         20) SpecialCH < 0.5 64   51.98 MM ( 0.14062 0.85938 ) *
##         21) SpecialCH > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
##       11) PriceDiff > 0.05 109  147.00 CH ( 0.59633 0.40367 ) *
##    3) LoyalCH > 0.5036 435  337.90 CH ( 0.86897 0.13103 )
##      6) LoyalCH < 0.764572 174  201.00 CH ( 0.73563 0.26437 )
##       12) ListPriceDiff < 0.235 72   99.81 MM ( 0.50000 0.50000 )
##         24) PctDiscMM < 0.196196 55   73.14 CH ( 0.61818 0.38182 ) *
##         25) PctDiscMM > 0.196196 17   12.32 MM ( 0.11765 0.88235 ) *
##       13) ListPriceDiff > 0.235 102   65.43 CH ( 0.90196 0.09804 ) *
##      7) LoyalCH > 0.764572 261   91.20 CH ( 0.95785 0.04215 ) *
```

```
Comments:
node number (8): the observations whose LoyalCH is smaller than
0.0356415 is classified from the upper branch into the node (8).
the number of observations fell in the node 8 is 59, and its deviance
is 10.14. the overall prediction for the node is "MM" and the fraction
f MM here is 0.98305.
```

(d) Create a plot of the tree, and interpret the results.

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




```
Comments:
the predictors used for the tree is LoyalCH, PriceDiff, SpecialCH,
ListPriceDiff, PctDiscMM.
among the predictors, LoyalCH located on the top is the most important variable
and the tree is consisting of 8 subtrees with 9 terminal nodes
```

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels.

```
tree.pred <- predict(tree.oj.train, test, type = "class")
conf.table <- table(tree.pred, test$Purchase)
print(conf.table)
```

```
##
## tree.pred  CH  MM
##        CH 160  38
##        MM   8  64
```

```
test.error <- 1-sum(diag(conf.table))/sum(conf.table)
print(paste("test error rate: ",round(test.error, 4)))
```

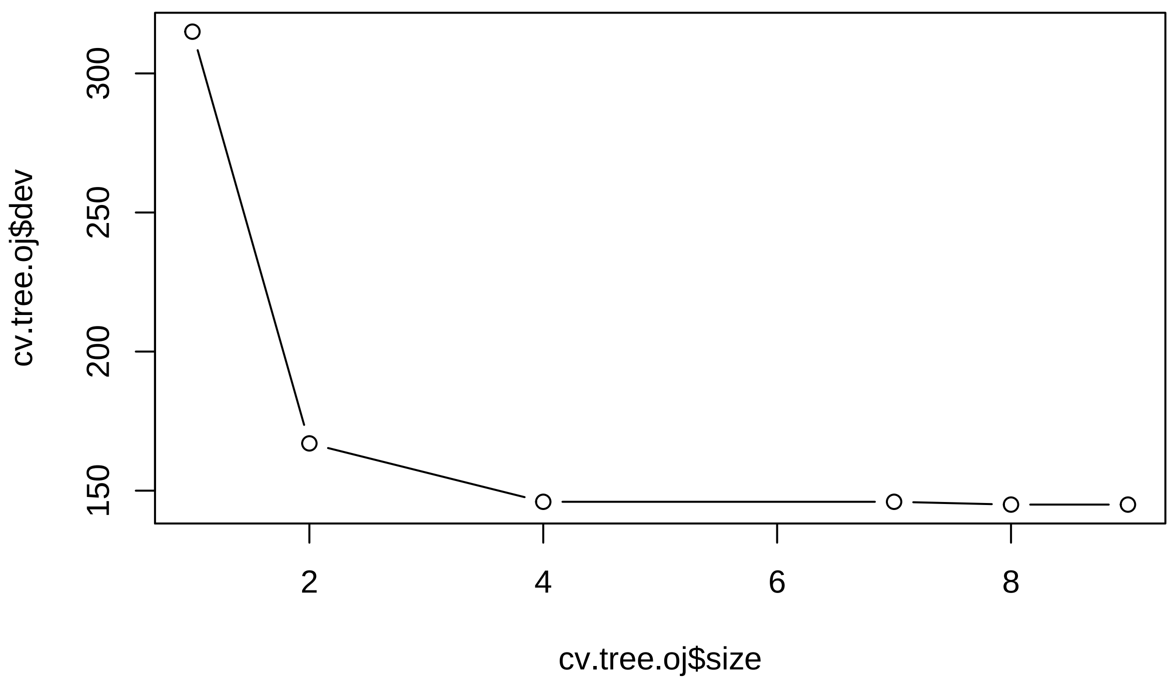
```
## [1] "test error rate:  0.1704"
```

(f) Determine the optimal tree size.

```
set.seed(1)
cv.tree.oj <- cv.tree(tree.oj.train, FUN = prune.misclass)
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

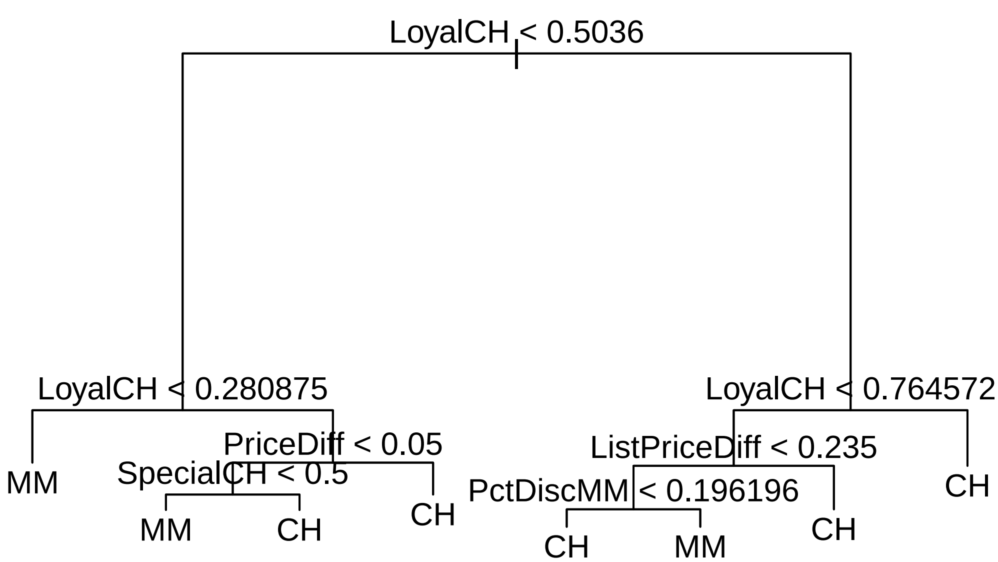
```
plot(cv.tree.oj$size, cv.tree.oj$dev, type = "b")
```




```
Comments:
based on the cross-validation error for each size, we can decide that
the best size(# terminal nodes) should be 8 or 9 because these have
the smallest CV error which is 145.
```

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation.

```r
## using 8 as the optimal size
prune.oj <- prune.misclass(tree.oj.train, best = 8)
plot(prune.oj)
text(prune.oj, pretty = 0)
```



(j) Compare the training error rates between the pruned and unpruned trees.

```r
summary(tree.oj.train)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## [5] "PctDiscMM"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7432 = 587.8 / 791
## Misclassification error rate: 0.1588 = 127 / 800
```

```r
summary(prune.oj)
```

```
##
## Classification tree:
## snip.tree(tree = tree.oj.train, nodes = 4L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## [5] "PctDiscMM"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7598 = 601.8 / 792
## Misclassification error rate: 0.1588 = 127 / 800
```

```
Comments:
training error rate of the unpruned model is 15.875%
training error rate of the pruned model is 15.875%
- the two models have the same training error rate
```

(k) Compare the test error rates between the pruned and unpruned trees.

```r
pred.pruned <- predict(prune.oj, test, type = "class")
conf.table.pruned <- table(pred.pruned, test$Purchase)
print(conf.table.pruned)
```

```
##
## pred.pruned  CH  MM
##          CH 160  38
##          MM   8  64
```

```r
test.error.pruned <- 1-sum(diag(conf.table.pruned))/sum(conf.table.pruned)

print(paste("test error rate of the unpruned model: ", test.error))
```

```
## [1] "test error rate of the unpruned model:  0.17037037037037"
```

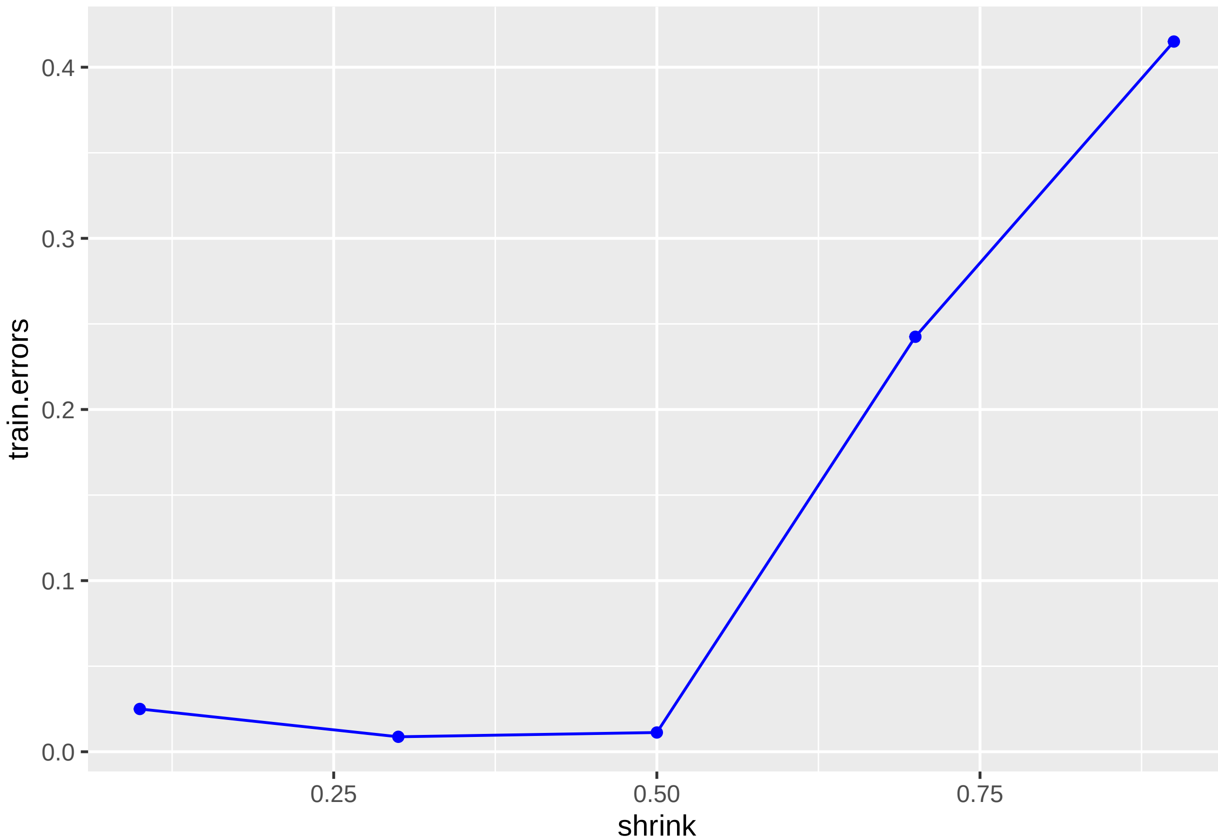
```r
print(paste("test error rate of the pruned model: ", test.error.pruned))
```

```
## [1] "test error rate of the pruned model:  0.17037037037037"
```

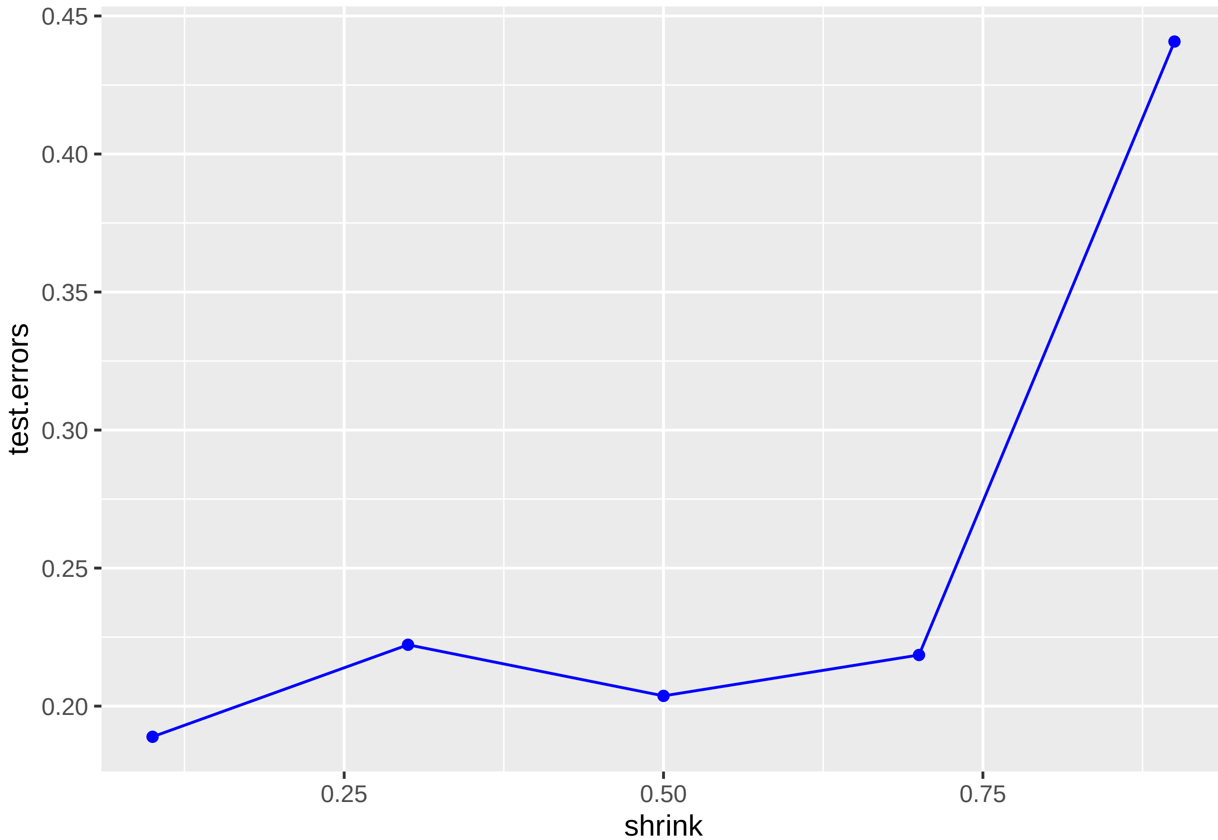
```
Comments:
the test error rates of the two models are exactly equal
```

(l) 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 error and test error on the y-axis.

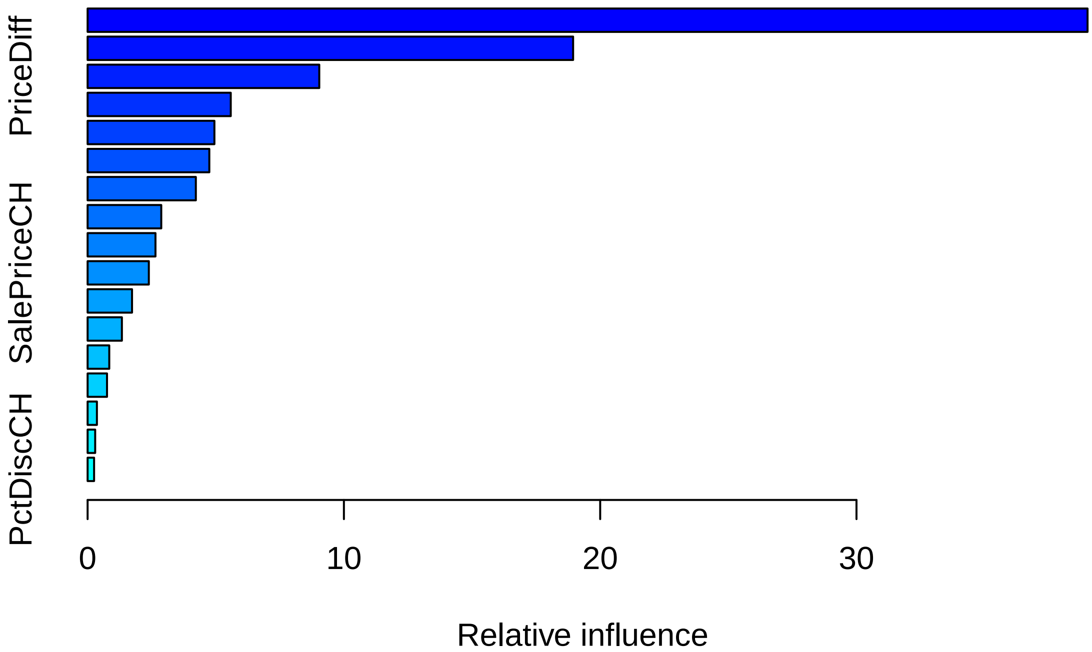
```r
train$Purchase <- ifelse(train$Purchase == "CH", 1, 0)
test$Purchase <- ifelse(test$Purchase == "CH", 1, 0)


## Plot with different shrinkage parameters and train errors
set.seed(1)
shrink <- c(0.1, 0.3, 0.5, 0.7, 0.9)
train.errors <- NULL

for(i in shrink){
  boost2 <- gbm(Purchase~., data = train, distribution = "bernoulli",
                n.trees = 1000, interaction.depth = 4, shrinkage=i)
  pred2 <- ifelse(predict(boost2, newdata = train,
                          n.trees=1000, type="response")>0.5,1,0)
  table2 <- table(pred2, train$Purchase)
  train.errors <- c(train.errors, 1-sum(diag(table2))/sum(table2))
}

train.err.df<- data.frame(train.errors, shrink)
ggplot(train.err.df, aes(x=shrink, y=train.errors))+
  geom_line(color = "blue")+
  geom_point(color = "blue")
```




```r
## Plot with different shrinkage parameters and test errors
set.seed(1)
test.errors <- NULL

for(i in shrink){
  boost1 <- gbm(Purchase~., data = train, distribution = "bernoulli",
                n.trees = 1000, interaction.depth = 4, shrinkage=i)
  pred1 <- ifelse(predict(boost1, newdata = test,
                          n.trees=1000, type="response")>0.5,1,0)
  table1 <- table(pred1, test$Purchase)
  test.errors <- c(test.errors, 1-sum(diag(table1))/sum(table1))
}

test.err.df<- data.frame(test.errors, shrink)
ggplot(test.err.df, aes(x=shrink, y=test.errors))+
  geom_line(color = "blue")+
  geom_point(color = "blue")
```




```
#results: shrinkage parameter 0.3 minimizes test error.

set.seed(1)
boost.oj.train <- gbm(Purchase~., data = train, distribution = "bernoulli",
                      n.trees = 1000, interaction.depth = 4, shrinkage = 0.3)


summary(boost.oj.train)
```




```
##                           var    rel.inf
## LoyalCH               LoyalCH 39.0157396
## WeekofPurchase WeekofPurchase 18.9430929
## PriceDiff           PriceDiff  9.0393249
## StoreID               StoreID  5.5850013
## ListPriceDiff   ListPriceDiff  4.9479441
## STORE                   STORE  4.7484284
## SalePriceMM       SalePriceMM  4.2228905
## PriceCH               PriceCH  2.8743455
## PriceMM               PriceMM  2.6483894
## SalePriceCH       SalePriceCH  2.3895021
## SpecialCH           SpecialCH  1.7332148
## DiscMM                 DiscMM  1.3380080
## PctDiscMM           PctDiscMM  0.8448488
## SpecialMM           SpecialMM  0.7576262
## DiscCH                 DiscCH  0.3626796
## Store7                 Store7  0.2956998
## PctDiscCH           PctDiscCH  0.2532640
```

```
pred.boost <- ifelse(predict(boost.oj.train, newdata = test,
                          n.trees=1000, type="response")>0.5,1,0)
table1 <- table(pred.boost, test$Purchase)
boost.test.error <- 1-sum(diag(table1))/sum(table1)

print(paste("test error of the boosting model with shrinkage parameter = 0.3 is: ", round(boost.test.er
```

```
## [1] "test error of the boosting model with shrinkage parameter = 0.3 is:  0.1963"
```

```
Comments:
the important predictors of this model are:
"LoyalCH" and "WeekofPurchase"
```

(m) Perform bagging on the training set and report the prediction performance on the test set.

```
set.seed(1)
split.train <- sample(1:nrow(oj), 800)
train <- oj[split.train,]
test <- oj[-split.train,]

set.seed(1)
bag.oj.train = randomForest(Purchase~., data = train, mtry=17, importance=TRUE)
pred.bag <- predict(bag.oj.train, newdata = test)
bag.table <- table(pred.bag, test$Purchase)
bag.error <- 1-sum(diag(bag.table))/sum(bag.table)
print(paste("test error of the bagging model: ", round(bag.error, 4)))
```

```
## [1] "test error of the bagging model:  0.1852"
```

```
bag.oj.train$importance
```

```
##                           CH           MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  1.449275e-02 0.0081761108         0.0119616206       38.3886426
## StoreID         2.813469e-03 0.0157406190         0.0078979001       12.6325113
## PriceCH         3.908277e-03 0.0031710145         0.0036595727        4.3656852
## PriceMM         2.065023e-03 0.0032540724         0.0025174091        4.4250917
## DiscCH         -6.707993e-04 0.0024988832         0.0005629286        1.7870508
## DiscMM          1.090828e-03 0.0032210479         0.0019057516        2.1431513
## SpecialCH       2.555862e-03 0.0043654998         0.0033002920        6.5637446
## SpecialMM      -5.084022e-05 0.0010224016         0.0003630224        2.6593366
## LoyalCH         1.370922e-01 0.2559065843         0.1835532385      227.1214620
## SalePriceMM     2.406628e-03 0.0184235916         0.0087588669       11.3677068
## SalePriceCH     4.229308e-03 0.0025857973         0.0035977121        5.7424289
## PriceDiff       2.459433e-02 0.0505692331         0.0346948871       31.1857003
## Store7          9.799625e-04 0.0003121969         0.0007186679        0.8956352
## PctDiscMM       1.631394e-03 0.0026749205         0.0020242202        2.4066397
## PctDiscCH      -2.783524e-04 0.0022054421         0.0006976463        2.2417075
## ListPriceDiff   1.205162e-02 0.0068240479         0.0099998522       13.7020010
## STORE           8.605124e-03 0.0110309564         0.0095168506        9.1696040
```

```
Comments:
test error is 18.52% and the top3 most important predictors are:
"LoyalCH", "WeekofPurchase", "PriceDiff"
```

(n) Perform random forest on the training set with $\sqrt{p}$ and $p/3$ predictors respectively and report the prediction performance on the test set.

```
# with sqrt(p) ntry (sqrt(p) is the default for random forest classification)
# sqrt(p) is the default value of randomForest function:
set.seed(1)
rf.oj.train <- randomForest(Purchase~., data = train, importance = TRUE)

rf.pred.sqrt <- predict(rf.oj.train, newdata = test)
rf.table.sqrt <- table(rf.pred.sqrt, test$Purchase)
rf.error.sqrt <- 1-sum(diag(rf.table.sqrt))/sum(rf.table.sqrt)
print(paste("test error of the rf model with sqrt(p) mtry: ", round(rf.error.sqrt, 4)))
```

```
## [1] "test error of the rf model with sqrt(p) mtry:  0.1704"
```

```r
# important predictors
print(rf.oj.train$importance)
```

```
##                           CH          MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  0.0086483865 0.015784775          0.011523662        28.946406
## StoreID         0.0096641735 0.031654548          0.018355206        21.499067
## PriceCH         0.0045708309 0.003595271          0.004283876         5.639893
## PriceMM         0.0019068217 0.007167926          0.004003358         6.272887
## DiscCH          0.0003710221 0.004770353          0.002105822         4.262464
## DiscMM          0.0042679863 0.009760813          0.006493291         5.355865
## SpecialCH       0.0009488070 0.004243129          0.002279976         3.445623
## SpecialMM      -0.0015845475 0.015044171          0.005037819         4.321306
## LoyalCH         0.1131614535 0.225523308          0.157423714       155.678184
## SalePriceMM     0.0078971280 0.018047829          0.011922103        11.362361
## SalePriceCH     0.0048710638 0.004073971          0.004501026         7.106994
## PriceDiff       0.0153258764 0.035382529          0.023292673        18.571671
## Store7          0.0077114010 0.013011886          0.009804308         6.045269
## PctDiscMM       0.0061440745 0.007567337          0.006711570         5.868678
## PctDiscCH       0.0011998844 0.004153033          0.002336332         4.298745
## ListPriceDiff   0.0088810210 0.013228920          0.010621118        10.781037
## STORE           0.0070591358 0.021300943          0.012796562        12.273028
```

```r
# with p/3 mtry
set.seed(1)
rf.oj.train2 <- randomForest(Purchase~., data = train, mtry=6, importance = TRUE)

rf.pred2<- predict(rf.oj.train2, newdata = test)
rf.table2 <- table(rf.pred2, test$Purchase)
rf.error2 <- 1-sum(diag(rf.table2))/sum(rf.table2)
print(paste("test error of the rf model with p/3 mtry: ", round(rf.error2, 4)))
```

```
## [1] "test error of the rf model with p/3 mtry:  0.1815"
```

```r
# important predictors
print(rf.oj.train2$importance)
```

```
##                           CH          MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  0.0110137766 0.015731401          0.012997396        32.928907
## StoreID         0.0101491974 0.025989866          0.016299691        20.998314
## PriceCH         0.0039803283 0.005347706          0.004561344         5.169630
## PriceMM         0.0014647245 0.006856471          0.003588800         6.060659
## DiscCH          0.0008635986 0.003304417          0.001826785         3.814710
## DiscMM          0.0043140140 0.004177192          0.004290142         4.197832
## SpecialCH       0.0014392566 0.003495084          0.002266185         4.080124
## SpecialMM      -0.0011149369 0.007385660          0.002238215         4.069948
## LoyalCH         0.1295831289 0.242840631          0.173901650       186.223445
## SalePriceMM     0.0061637415 0.017952261          0.010753337        12.422317
## SalePriceCH     0.0058933588 0.006389003          0.006061505         7.957959
## PriceDiff       0.0177050757 0.037770237          0.025669676        21.000941
## Store7          0.0062420816 0.009031380          0.007355507         5.453113
```

```
## PctDiscMM       0.0045882050 0.006137642          0.005233079          5.227907
## PctDiscCH       0.0002484615 0.004051293          0.001751392          3.804181
## ListPriceDiff   0.0073224034 0.010966501          0.008810964         11.517964
## STORE           0.0062344958 0.017638572          0.010781193         10.468506
```

```
Comments:
* for mtry = sqrt(p):
  test error is 17.04% and the top3 the most important predictors are:
  "LoyalCH", "WeekofPurchase", "StoreID"

* for mtry = p/3:
  test error is 18.15% and the top3 the most important predictors are:
  "LoyalCH", "WeekofPurchase", "PriceDiff"
```

(o) Compare the above models.

1) test error of Pruned Tree: 0.1704 the top predictor: "LoyalCH"
2) test error of the Boosting model with shrinkage parameter = 0.3: 0.1963 important predictors: "LoyalCH", "WeekofPurchase"
3) test error of the Bagging model: 0.1852" important predictors: "LoyalCH", "WeekofPurchase"
4) test error of the Random Forest with mtry = sqrt(p): 0.1704 important predictors are: "LoyalCH", "WeekofPurchase"
5) test error of the Random Forest with mtry = p/3: 0.1815 important predictors are: "LoyalCH", "WeekofPurchase"

Results: the Pruned Tree model and the Random Forest model with mtry=3/p produce the lowest test errors and their importat predictors are the same