RSM8512 Assignment - NTree and SVM

Yanbing Chen 1009958752 2023/11/29

Question 1 [20 marks]

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 quantitative variable.

(a) Split the data set into a training set and a test set.

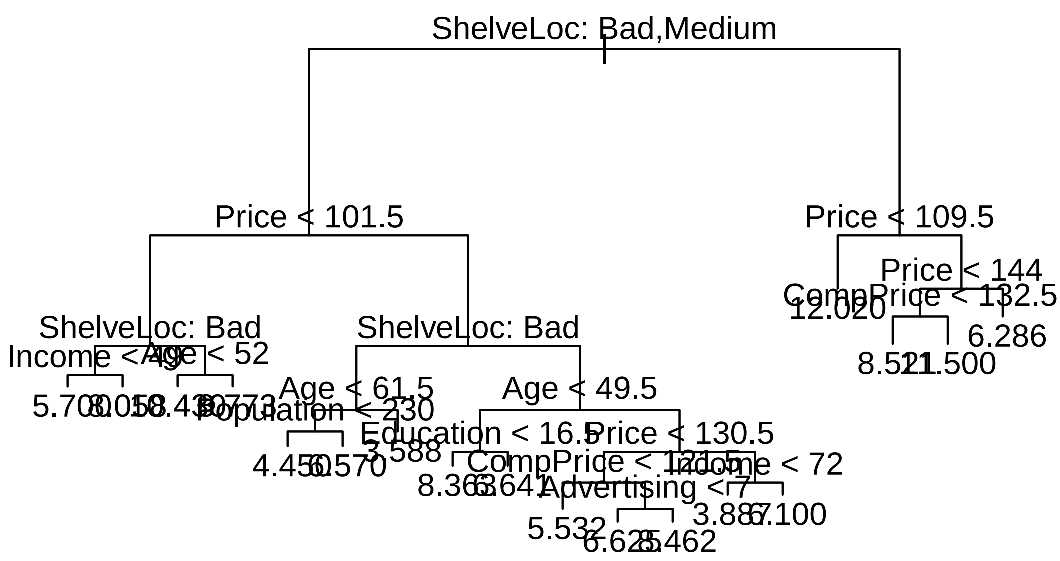
```
data("Carseats")
set.seed(123)

train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
Carseats.train = Carseats[train,]
Carseats.test = Carseats[-train,]
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

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

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "Age"         "Population"
## [6] "Education"   "CompPrice"   "Advertising"
## Number of terminal nodes:  18
## Residual mean deviance:  2.132 = 388.1 / 182
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.08000 -0.92870  0.06244  0.00000  0.87020  3.71700
```

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



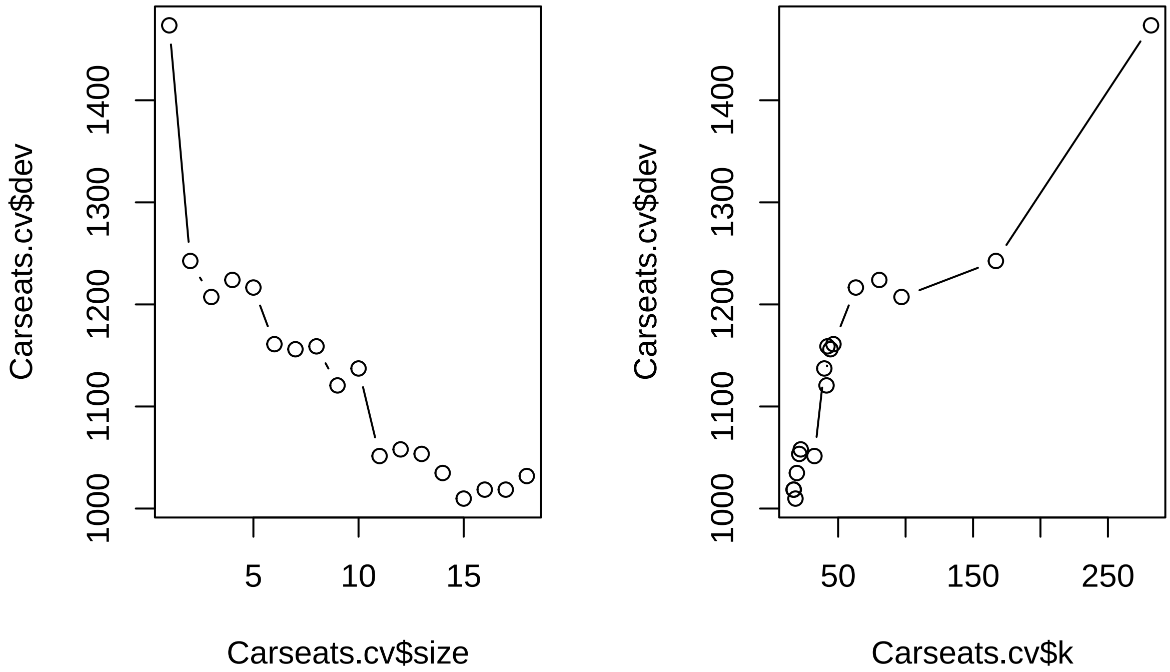

```r
pred.Carseats = predict(Carseats.tree, Carseats.test)
mean((Carseats.test$Sales - pred.Carseats)^2)
```

```
## [1] 4.395357
```

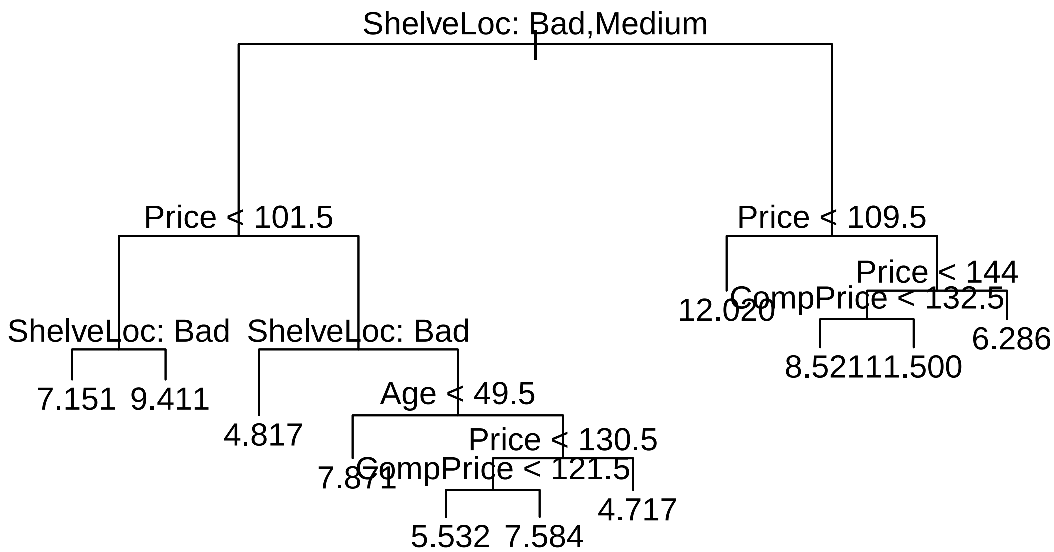
The test MSE is 4.395357.

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

```r
Carseats.cv = cv.tree(Carseats.tree, FUN = prune.tree)
par(mfrow = c(1,2))
plot(Carseats.cv$size, Carseats.cv$dev, type = "b")
plot(Carseats.cv$k, Carseats.cv$dev, type = "b")
```




```r
# Best size = 11
Carseats.pruned = prune.tree(Carseats.tree, best = 11)
par(mfrow= c(1,1))
plot(Carseats.pruned)
text(Carseats.pruned, pretty = 0)
```




```
pred.pruned = predict(Carseats.pruned, Carseats.test)
mean((Carseats.test$Sales - pred.pruned)^2)
```

```
## [1] 4.646409
```

After pruning the tree, the test MSE increase to 4.646.

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

```
Carseats.bag = randomForest(Sales ~., data = Carseats.train, mtry = 10, ntree = 500, importance
pred.Carseats = predict(Carseats.bag, Carseats.test)
mean((Carseats.test$Sales - pred.Carseats)^2)
```

```
## [1] 2.706945
```

The test MSE is 2.731395

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

```
##                %IncMSE IncNodePurity
## CompPrice   20.45893952    163.315084
## Income       5.99352172     88.626184
## Advertising  6.70900949     73.007073
## Population  -1.84004720     53.079505
## Price       46.01586429    395.251820
## ShelveLoc   49.31816789    391.948958
## Age         17.74691675    171.659574
## Education    2.98753578     57.308595
## Urban        0.04864498      7.721022
## US           0.05207339      6.011265
```

Using bagging approach can improve the test MSE to 2.71585. In addition, from te result above we can see `Price`, `ShelveLoc` and `Age` are three most important factors.

(e) Use random forests to analyze this data. 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.

```r
Carseats.rf = randomForest(Sales ~ ., data = Carseats.train, mtry = 5, ntree = 500,
    importance = T)
pred.rf = predict(Carseats.rf, Carseats.test)
mean((Carseats.test$Sales - pred.rf)^2)
```

```
## [1] 3.092076
```

The random Forest Model worsen the test MSE to 3.074. Changing m varies test MSE between 2.73 to 3.07.

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

```
##               %IncMSE IncNodePurity
## CompPrice   12.8153277    146.261391
## Income       7.0766835    111.252330
## Advertising  6.2121481     89.901889
## Population  -1.2975016     79.429067
```

```
## Price        35.5344924    331.725876
## ShelveLoc    41.3002362    335.894363
## Age          17.9183484    204.422771
## Education    -0.6312320     64.334408
## Urban        -1.0644618     10.562869
## US           -0.0673826      9.792536
```

From the result above we can see, the three most important factors are `ShelveLoc`, `Price` and `Age`.

Question 2 [24 marks]

This problem involves the OJ data set which is part of the ISLR2 package.

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

```
set.seed(1234)
data(OJ)
train = sample(dim(OJ)[1], 800)
OJ_train = OJ[train,]
OJ_test = OJ[-train,]
```

(b) Fit a support vector classifier to the training data using $cost = 0.01$, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
library(e1071)
svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ_train, cost = 0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.01
##
## Number of Support Vectors:  437
##
##  ( 219 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support Vector classifier creates 437 support vectors out of 800 training points. Out of these, 219 belong to level CH and remaining 218 belong to level MM.

(c) What are the training and test error rates?

```
train.pred = predict(svm.linear, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 426  57
##   MM  78 239
```

```
(78+57)/(426+57+78+239)
```

```
## [1] 0.16875
```

The training error rate is 0.16875.

```
test.pred = predict(svm.linear, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 151  19
##   MM  24  76
```

```r
(19+24)/(151+19+24+76)
```

```
## [1] 0.1592593
```

The test error rate is 0.1592593.

(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```r
set.seed(1234)
tune.out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "linear", ranges = list(cost = 10^
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        cost
##  0.05623413
##
## - best performance: 0.16875
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17625 0.03304563
## 2   0.01778279 0.17375 0.03653860
## 3   0.03162278 0.17250 0.03050501
## 4   0.05623413 0.16875 0.03186887
## 5   0.10000000 0.17375 0.03701070
## 6   0.17782794 0.17625 0.03884174
## 7   0.31622777 0.17250 0.03476109
## 8   0.56234133 0.17125 0.03335936
## 9   1.00000000 0.17000 0.03238227
## 10  1.77827941 0.17125 0.03775377
## 11  3.16227766 0.17125 0.03387579
## 12  5.62341325 0.17125 0.03283481
## 13 10.00000000 0.17125 0.03283481
```

The optimal cost is 0.05623.

(e) Compute the training and test error rates using this new value for cost.

```
svm.linear =  svm(Purchase ~ ., kernel = "linear", data = OJ_train, cost = tune.out$best.parame
train.pred = predict(svm.linear, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 426  57
##   MM  75 242
```

```
(57+75)/(426+57+75+242)
```

```
## [1] 0.165
```

```
test.pred = predict(svm.linear, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 152  18
##   MM  24  76
```

```
(18+24)/(152+18+24+76)
```

```
## [1] 0.1555556
```

After using the best cost, the test error rate decreases to 0.1555556, and the training error rate decreases to 0.165.

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```r
set.seed(1234)
svm.radial = svm(Purchase ~ ., data = OJ_train, kernel = "radial")
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  375
##
##  ( 188 187 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred = predict(svm.radial, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 442  41
##   MM  79 238
```

```r
(41+79)/(442+238+41+79)
```

```
## [1] 0.15
```

```r
test.pred = predict(svm.radial, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 154  16
##   MM  26  74
```

```r
(16+26)/(154+16+26+74)
```

```
## [1] 0.1555556
```

```r
set.seed(1234)
tune.out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "radial", ranges = list(cost = 10^
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.5623413
##
## - best performance: 0.1825
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39625 0.05466120
## 2   0.01778279 0.39625 0.05466120
## 3   0.03162278 0.34375 0.07665987
## 4   0.05623413 0.21000 0.04816061
## 5   0.10000000 0.20625 0.05212498
## 6   0.17782794 0.19625 0.04752558
## 7   0.31622777 0.18500 0.04362084
## 8   0.56234133 0.18250 0.04048319
```

```
## 9    1.00000000 0.18875 0.04267529
## 10   1.77827941 0.19000 0.03574602
## 11   3.16227766 0.18750 0.02763854
## 12   5.62341325 0.19125 0.03283481
## 13 10.00000000 0.20000 0.03632416
```

The radial basis kernel with default gamma creates 375 support vectors, out of which, 188 belong to level CH and remaining 187 belong to level MM. The classifier has a training error of 15% and % and a test error of 15.6%. We now use cross validation to find optimal gamma.

```
set.seed(2345)
tune.out = tune(svm, Purchase ~., data = OJ_train, kernel = "radial", ranges = list(cost = 10^s
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.1778279
##
## - best performance: 0.17625
##
## - Detailed performance results:
##            cost   error dispersion
## 1    0.01000000 0.39625 0.07684083
## 2    0.01778279 0.39625 0.07684083
## 3    0.03162278 0.35000 0.10290908
## 4    0.05623413 0.20250 0.04594683
## 5    0.10000000 0.18625 0.03030516
## 6    0.17782794 0.17625 0.03557562
## 7    0.31622777 0.18750 0.04082483
## 8    0.56234133 0.18000 0.03641962
## 9    1.00000000 0.17750 0.03425801
## 10   1.77827941 0.18125 0.03019037
## 11   3.16227766 0.18375 0.02766993
```

```
## 12  5.62341325 0.18500 0.03050501
## 13 10.00000000 0.18875 0.02913689
```

```
svm.radial = svm(Purchase ~ ., data = OJ_train, kernel = "radial", cost = tune.out$best.paramet
train.pred = predict(svm.radial, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 435  48
##   MM  79 238
```

```
(48+79)/(435+48+79+239)
```

```
## [1] 0.1585518
```

```
test.pred = predict(svm.radial, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 153  17
##   MM  24  76
```

```
(17+24)/(153+17+24+76)
```

```
## [1] 0.1518519
```

Tuning does increase the training error but slightly decrease the test error.

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree = 2.

```
set.seed(3456)
svm.poly = svm(Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  475
##
##  ( 243 232 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred = predict(svm.poly, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 446  37
##   MM 114 203
```

```
(37+114)/(446+37+114+203)
```

```
## [1] 0.18875
```

```
test.pred = predict(svm.poly, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
```

```
##       CH  MM
##   CH 157  13
##   MM  30  70
```

```r
(13+30)/(157+13+30+70)
```

```
## [1] 0.1592593
```

According to the result above, the polynomial kernel produces 475 support vectors, out of which, 243 belong to level CH and remaining 232 belong to level MM. This kernel produces a train error of 18.9% and a test error of 15.93%, which are higher than the errors produces by radial kernel and the errors produced by linear kernel.

```r
set.seed(3456)
tune.out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2,
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.18875
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39250 0.03689324
## 2   0.01778279 0.38000 0.02898755
## 3   0.03162278 0.37250 0.03476109
## 4   0.05623413 0.35500 0.03395258
## 5   0.10000000 0.33000 0.02776389
## 6   0.17782794 0.26375 0.03251602
## 7   0.31622777 0.21500 0.02993047
```

```
## 8    0.56234133 0.21750 0.03184162
## 9    1.00000000 0.21000 0.02751262
## 10   1.77827941 0.19625 0.02638523
## 11   3.16227766 0.19250 0.02713137
## 12   5.62341325 0.19000 0.02415229
## 13  10.00000000 0.18875 0.03087272
```

```
svm.poly = svm(Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2, cost = tune.out$best
train.pred = predict(svm.poly, OJ_train)
table(OJ_train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 443  40
##   MM  85 232
```

```
(40+85)/(443+40+85+232)
```

```
## [1] 0.15625
```

```
test.pred = predict(svm.poly, OJ_test)
table(OJ_test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 155  15
##   MM  27  73
```

```
(15+27)/(155+15+27+73)
```

```
## [1] 0.1555556
```

Tuning reduces the training error to 15.63%, and the test error to 15.56% which is better than before.

(h) Overall, which approach seems to give the best results on this data?

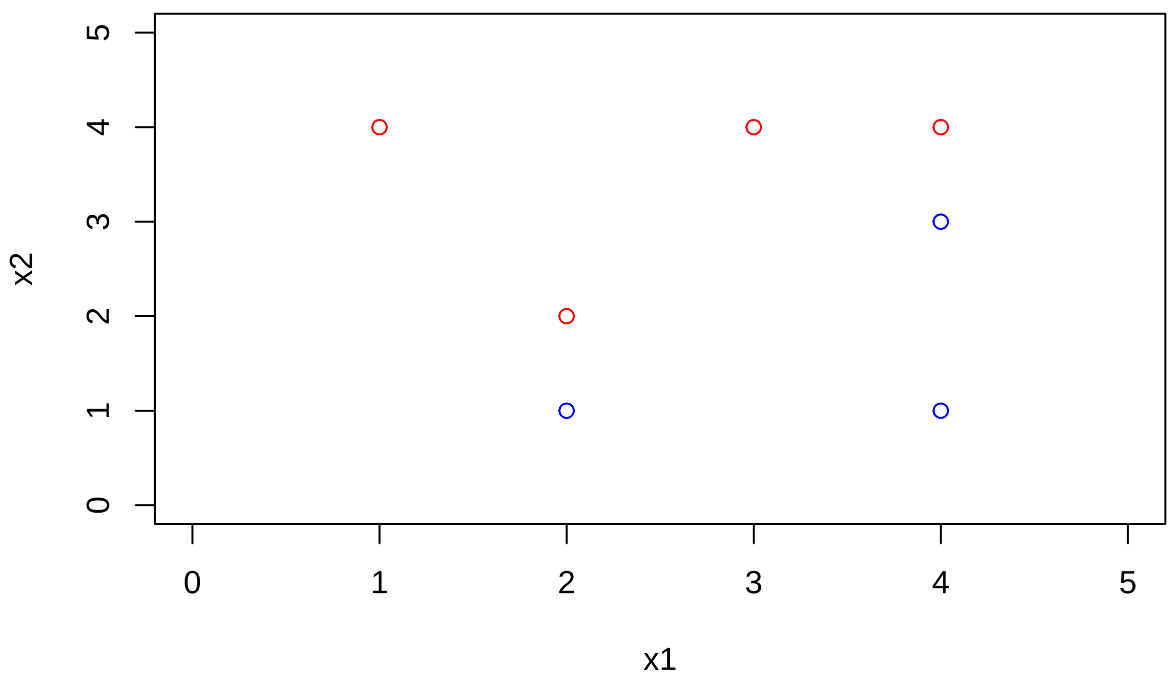
Answer: To summarize, radial basis kernel seems to be the best on both train and test data.

Bonus Question 3 [8 marks]

Here we explore the maximal margin classifier on a toy data set.

(a) We are given $n = 7$ observations in $p = 2$ dimensions. FOr each observation, there is an associated class label. Sketch the observations.

```
x1 = c(3,2,4,1,2,4,4)
x2 = c(4,2,4,4,1,3,1)
Y = c("red","red","red","red", "blue","blue","blue")
plot(x1,x2, col = Y, xlim = c(0,5), ylim = c(0,5))
```

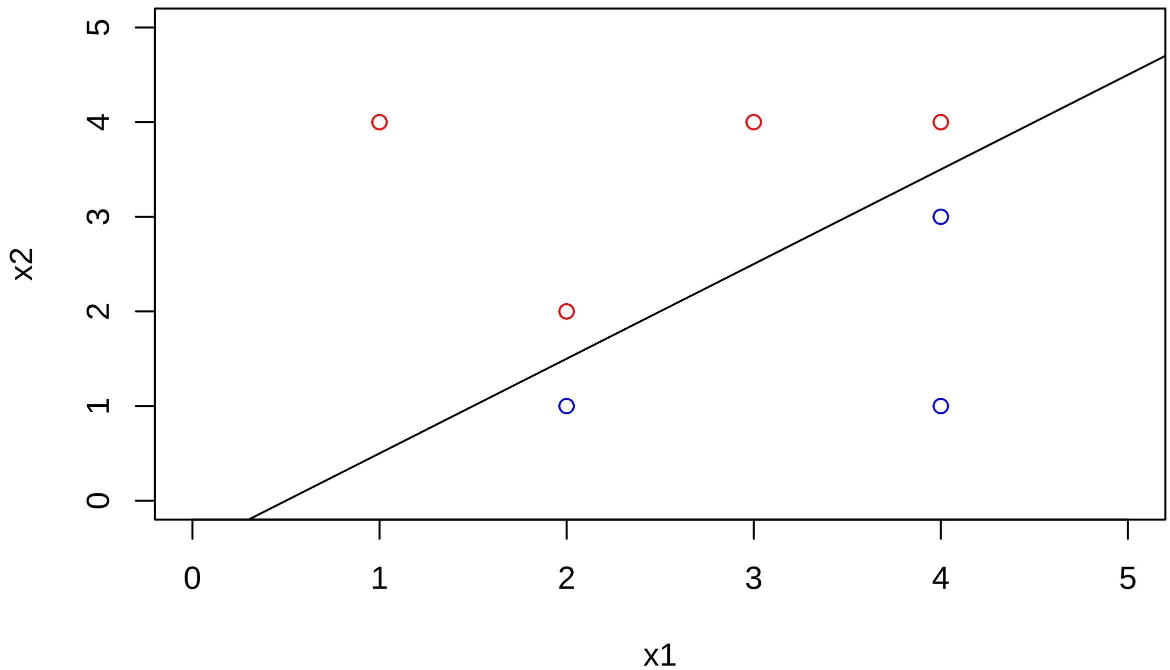


(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

Answer: The maximal classfier can be calculated by observations #2, #3, #5 and $6.

By transforming their coordinate, we get $(2, 1.5), (4, 3.5)$, so we can get $b = (y_2 - y_1)/(x_2 - x_1) = (3.5 - 1.5)/(4 - 2) = 1$, $a = x2 - x1 = 1.5 - 2 = -0.5$.

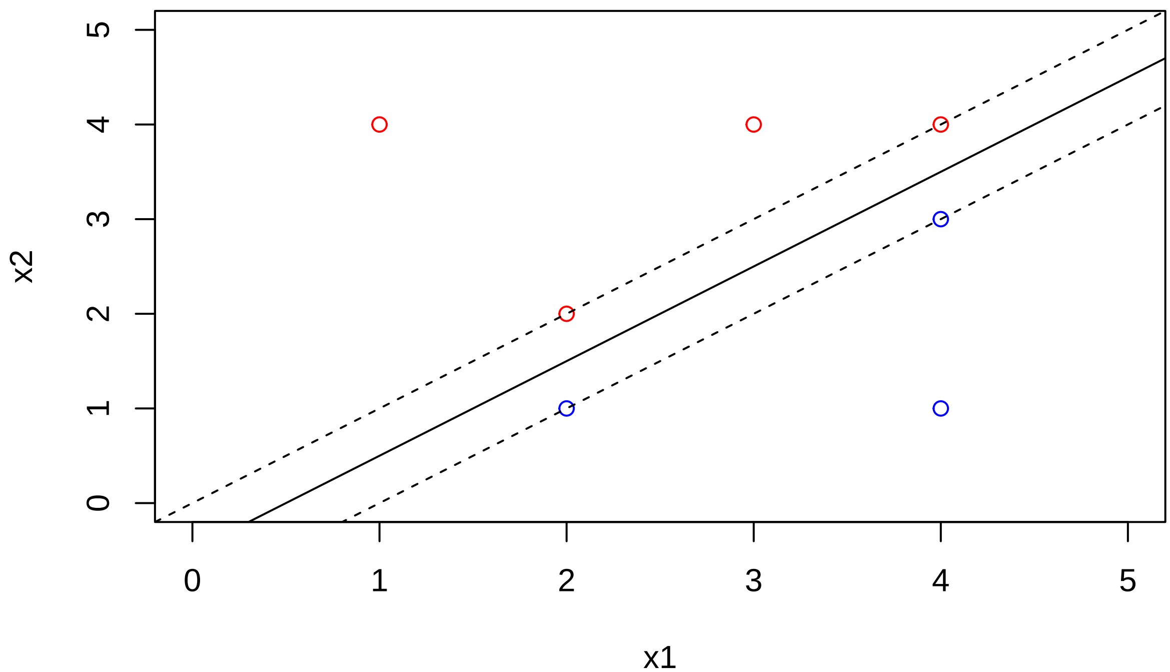
```
plot(x1,x2, col = Y, xlim = c(0,5), ylim = c(0,5))
abline(-0.5,1)
```



(c) **Answer:** According to the result gain above, we know $\beta_0 = 0.5$, $\beta_1 = -1$, $\beta_2 = 1$, So we have $0.5 - X_1 + X_2 > 0$.

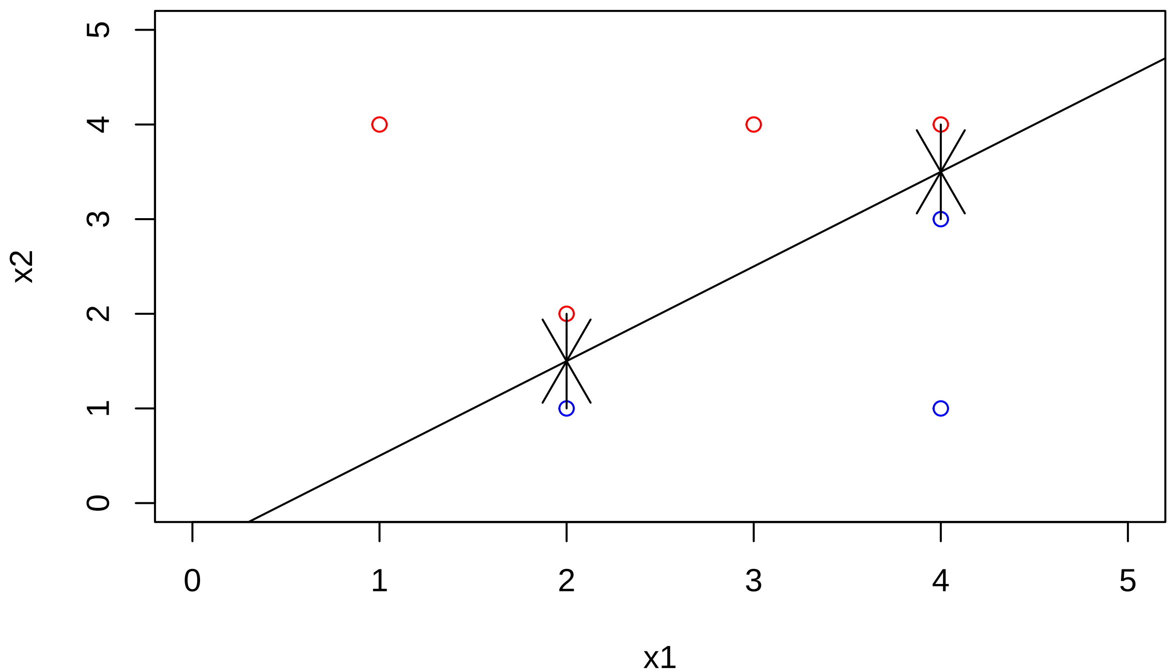
(d) On you sketch, indicate the margin for the maximal margin hyperplane.

```
plot(x1, x2, col = Y, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



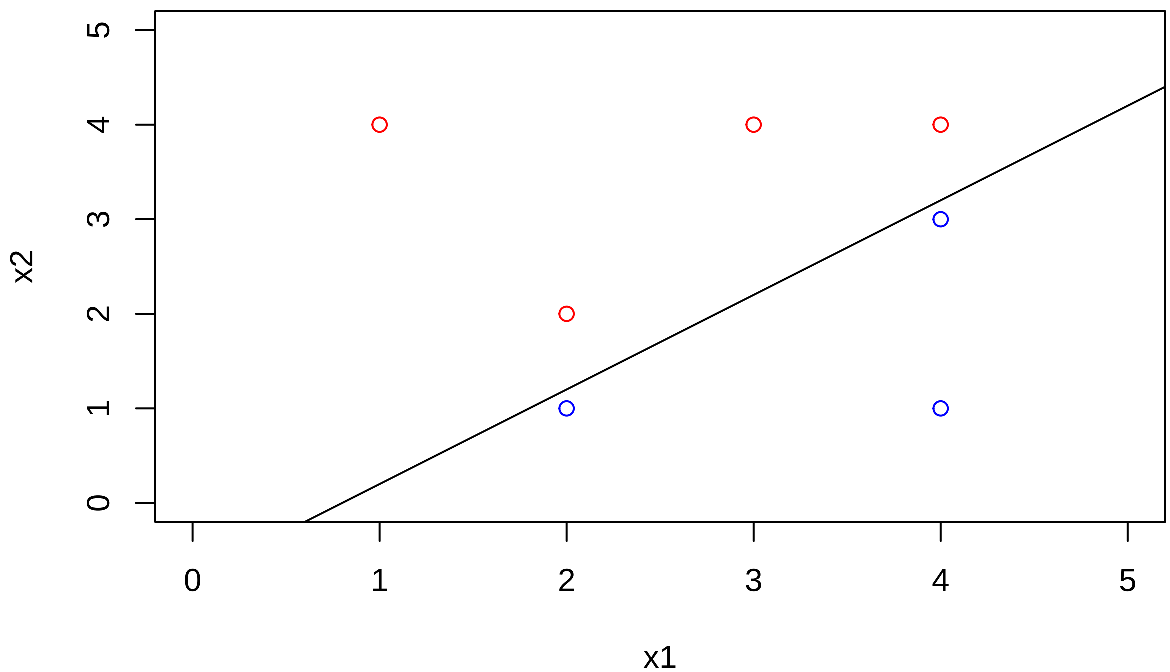
(e) Indicate the support vectors for the amximal margin classifier.

```
plot(x1, x2, col = Y, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
arrows(2, 1, 2, 1.5)
arrows(2, 2, 2, 1.5)
arrows(4, 4, 4, 3.5)
arrows(4, 3, 4, 3.5)
```



(g) Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

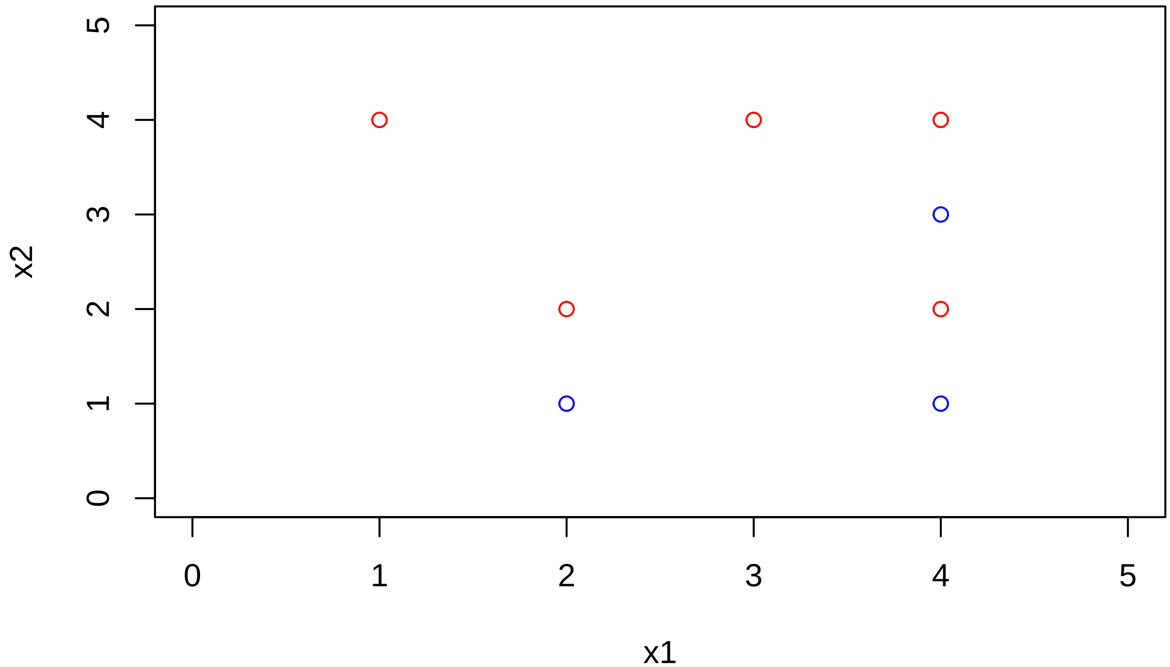
```
plot(x1, x2, col = Y, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.8,1)
```



The equation is $-0.8 - X_1 + X_2 > 0$.

(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

```r
plot(x1, x2, col = Y, xlim = c(0, 5), ylim = c(0, 5))
points(c(4), c(2), col = c("red"))
```



x2
0
1
2
3
4
5
x1
0
1
2
3
4
5