# Assignment 3 Solution

**Question 1**

(a) The smallest training RSS will be for the model with best subset approach. This is because the model will be chosen after considering all the possible models with k parameters for best subset. This is not true for either backward stepwise or forward stepwise.
(b) Best subset selection may have the smallest test MSE because it considers more models then the other methods. However, the other models might have better luck picking a model that fits the test data better.
(c) i. True. ii. True. iii. False. iv. False. v. False.

**Question 2**

(a) iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.
(b) iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

**Question 3**

(a) iv. Steadily decrease

With increase in s we are making the model more and more flexible as the restriction on $\beta$ is reducing. This will lead to decreased RSS.

(b) ii. Decrease initially, and then eventually start increasing in a U shape

As the model is becoming more and more flexible, it will reduce first and then start increasing when overfitting will start

(c) iii. Steadily increase

Variance steadily increase with the increase in model flexibility

(d) iv. Steadily decrease

Bias decreases with the increase in the model flexibility

**Question 4**

(a) iii. Steadily increase

With increase in $\lambda$ we are making the model less and less flexible as the restriction on $\beta$ is increasing. This will lead to increased RSS.

(b) ii. Decrease initially, and then eventually start increasing in a U shape

As the model is becoming less and less flexible, it will decrease first and then start increasing when overfitting will start

(c) iv. Steadily decrease

Variance steadily decreases with the decrease in model flexibility

(d) iii. Steadily increase

Bias increases with the decrease in the model flexibility

## Question 5

(a)

```
library(ISLR)
set.seed(11)
train.size = dim(College)[1] / 2
train = sample(1:dim(College)[1], train.size)
test = -train
College.train = College[train, ]
College.test = College[test, ]
```

(b)

```
lm.fit = lm(Apps~., data=College.train)
lm.pred = predict(lm.fit, College.test)
mean((College.test[, "Apps"] - lm.pred)^2)
```

```
## [1] 1538442
```

Test RSS is 1538442

(c)

```
library(glmnet)
train.mat = model.matrix(Apps~., data=College.train)
test.mat = model.matrix(Apps~., data=College.test)
grid = 10 ^ seq(4, -2, length=100)
mod.ridge = cv.glmnet(train.mat, College.train[, "Apps"], alpha=0,
lambda=grid, thresh=1e-12)
lambda.best = mod.ridge$lambda.min
ridge.pred = predict(mod.ridge, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - ridge.pred)^2)
```

```
## [1] 1608859
```

Test RSS is slightly higher that OLS, 1608859 with lambda.best = 18.74.

(d)

```
mod.lasso = cv.glmnet(train.mat, College.train[, "Apps"], alpha=1,
lambda=grid, thresh=1e-12)
lambda.best = mod.lasso$lambda.min
lasso.pred = predict(mod.lasso, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - lasso.pred)^2)
```

```
## [1] 1635280
```

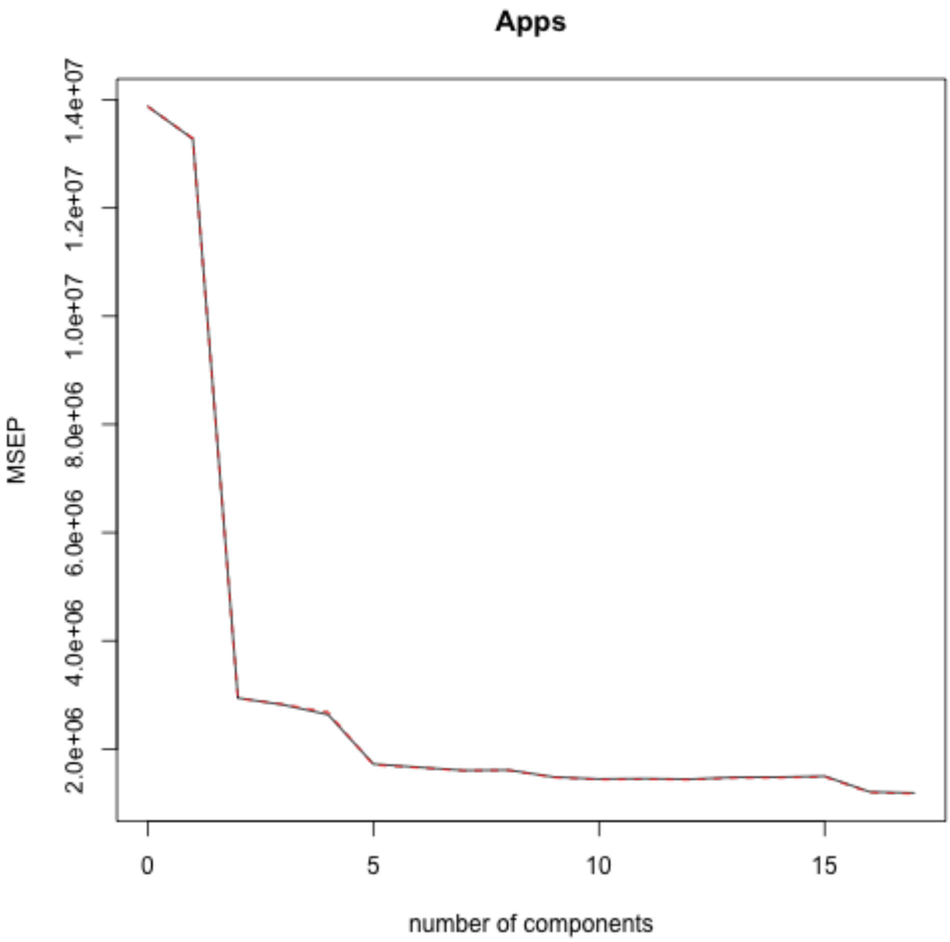
```
mod.lasso = glmnet(model.matrix(Apps~., data=College), College[, "Apps"],
alpha=1)
predict(mod.lasso, s=lambda.best, type="coefficients")
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -6.038e+02
## (Intercept)  .
## PrivateYes  -4.235e+02
## Accept       1.455e+00
## Enroll      -2.004e-01
## Top10perc    3.368e+01
## Top25perc   -2.403e+00
## F.Undergrad  .
## P.Undergrad  2.086e-02
## Outstate    -5.782e-02
## Room.Board   1.246e-01
## Books        .
## Personal     1.833e-05
## PhD         -5.601e+00
## Terminal    -3.314e+00
## S.F.Ratio    4.479e+00
## perc.alumni -9.797e-01
## Expend       6.968e-02
## Grad.Rate    5.160e+00
```

Again, Test RSS is slightly higher that OLS, 1635280 with lambda.best = 21.54.

(e)

```
library(pls)
pcr.fit = pcr(Apps~., data=College.train, scale=T, validation="CV")
validationplot(pcr.fit, val.type="MSEP")
```




```
pcr.pred = predict(pcr.fit, College.test, ncomp=10)
mean((College.test[, "Apps"] - data.frame(pcr.pred))^2)
```

```
## [1] 3014496
```

Test RSS for PCR is about 3014496, with M = 10.

**Question 6**

(a) With the majority vote approach, we classify X as Red as it is the most commonly occurring class among the 10 predictions (6 for Red vs 4 for Green).

(b) With the average probability approach, we classify X as Green as the average of the 10 probabilities is 0.45.

### Question 7

(a)

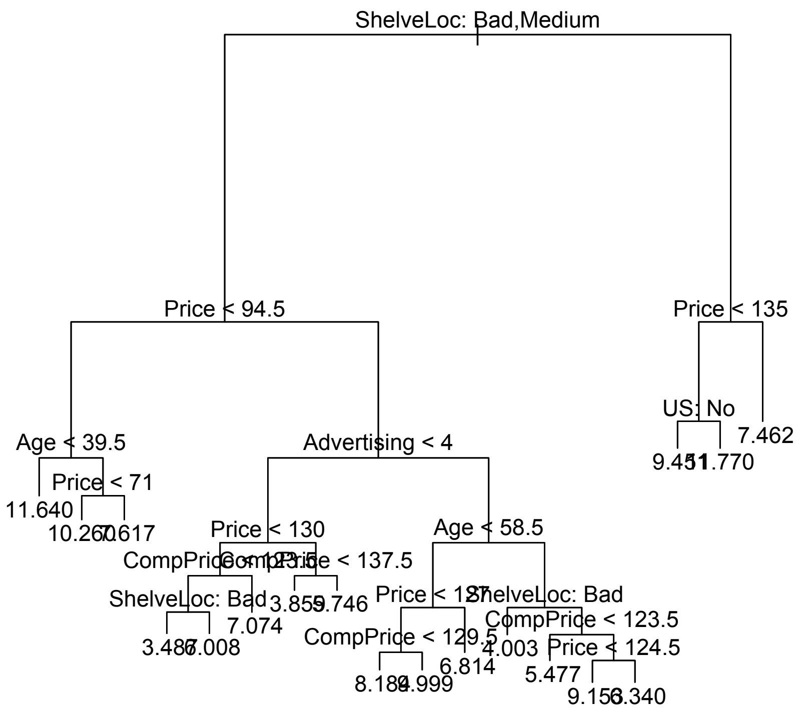
```
library(ISLR)
attach(Carseats)
set.seed(1)
train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

(b)

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

```
##  Regression tree:
##  tree(formula = Sales ~ ., data = Carseats.train)
##  Variables actually used in tree construction:
##  [1] "ShelveLoc"   "Price"       "Age"          "Advertising" "CompPrice"
"US"
##  Number of terminal nodes:  18
##  Residual mean deviance:  2.167 = 394.3 / 182
##  Distribution of residuals:
##        Min.    1st Qu.   Median      Mean   3rd Qu.      Max.
## -3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
```

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



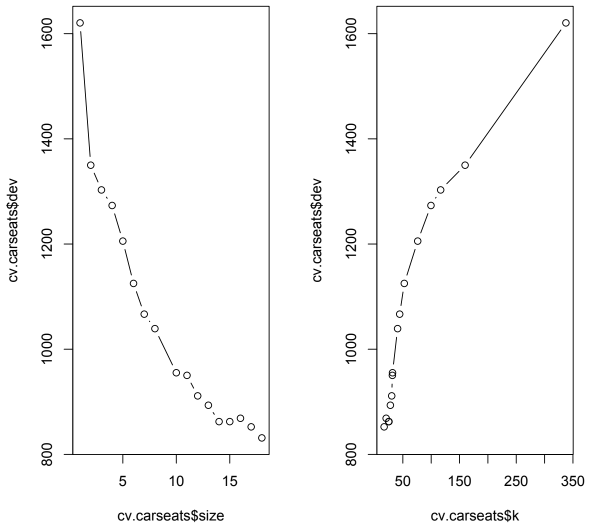

```
pred.carseats = predict(tree.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.carseats)^2)
```

```
## [1] 4.922039
```

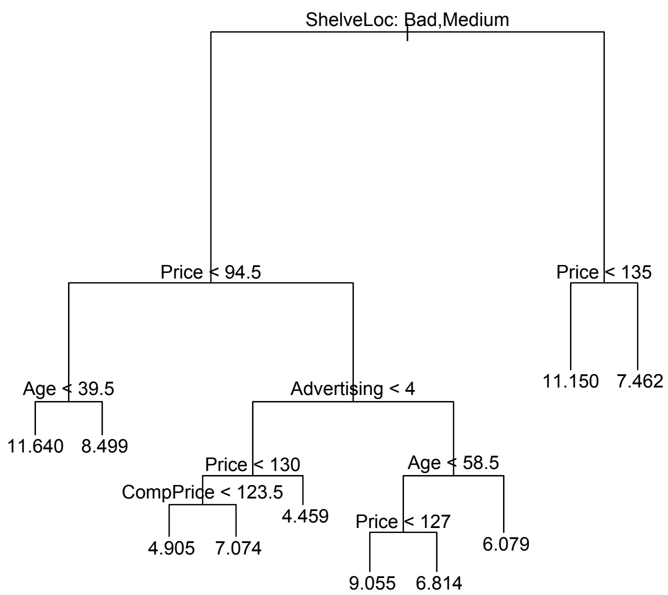
The test MSE is about 4.92.

(c)

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




```
# Best size = 9
pruned.carseats = prune.tree(tree.carseats, best = 9)
par(mfrow = c(1, 1))
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)
```




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

```
## [1] 4.918134
```

Pruning the tree in this case decreases the test MSE to 4.918

(d)

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

```
## [1] 2.657296
```

```
importance(bag.carseats)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   23.07909904    171.185734
## Income       2.82081527     94.079825
## Advertising 11.43295625     99.098941
## Population  -3.92119532     59.818905
## Price       54.24314632    505.887016
## ShelveLoc   46.26912996    361.962753
## Age         14.24992212    159.740422
## Education   -0.07662320     46.738585
## Urban        0.08530119      8.453749
## US           4.34349223     15.157608
```

Bagging improves the test MSE to 2.657296. We also see that Price,  ShelveLoc, CompPrice and Age are the most important predictors of Sale.

(e)

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

```
## [1] 2.701665
```

```
importance(rf.carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice   19.8160444     162.73603
## Income       2.8940268     106.96093
## Advertising 11.6799573     106.30923
## Population  -1.6998805      79.04937
## Price       46.3454015     448.33554
## ShelveLoc   40.4412189     334.33610
## Age         12.5440659     169.06125
## Education    1.0762096      55.87510
## Urban        0.5703583      13.21963
## US           5.8799999      25.59797
```

In this case, random forest worsens the MSE on test set to 2.701665. Changing m varies test MSE between 2.6 to 3. We again see that Price, ShelveLoc, CompPrice and Age are three the important predictors of Sale.

### Question 8

(a)

```
library(ISLR)
set.seed(9004)
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

(b)

```
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: 435
##
## ( 219 216 )
##
##
## Number of Classes: 2
##
## Levels:
## CH MM
```

Support vector classifier creates 435 support vectors out of 800 training points. Out of these, 219 belong to level CH and remaining 216 belong to level MM.

(c)

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

```
##   train.pred
##      CH  MM
## CH 420  65
## MM  75 240
```

```
(65+75)/(420+75+65+240)
```

```
## [1] 0.175
```

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

```
##   test.pred
##      CH MM
## CH 142 19
## MM  29 80
```

```
(33+15)/(153+33+15+69)
```

```
## [1] 0.1778
```

The training error rate is 17.5% and test error rate is about 17.8%.

(d)

```
set.seed(1554)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", 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
##  0.3162278
##
## - best performance: 0.17125
##
## -  Detailed performance results:
##                 cost       error    dispersion
## 1     0.01000000  0.17750  0.06635343
## 2     0.01778279  0.17750  0.05916080
## 3     0.03162278  0.17500  0.06095308
## 4     0.05623413  0.17375  0.06755913
## 5     0.10000000  0.17625  0.06755913
## 6     0.17782794  0.17625  0.06573569
## 7     0.31622777  0.17125  0.06483151
## 8     0.56234133  0.17375  0.06573569
## 9     1.00000000  0.17250  0.06258328
## 10   1.77827941  0.17500  0.06997023
## 11   3.16227766  0.17250  0.06661456
## 12   5.62341325  0.17625  0.07155272
## 13 10.00000000  0.17875  0.07072295
```

Tuning shows that optimal cost is 0.3162

(e)

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

```
##     train.pred
##       CH   MM
## CH   423   62
## MM    71  244
```

```
(62+71)/(423+62+71+244)
```

```
## [1] 0.16625
```

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

```
##     test.pred
##       CH  MM
## CH   155  13
## MM    29  73
```

```
(29+13)/(155+29+13+73)
```

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

The training error decreases to 16.625%, and test error decreases to 15.56% by using best cost.

(f)

```
set.seed(410)
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: 373
##
## ( 188 185 )
##
##
## Number of Classes: 2
##
## Levels:
## CH MM
```

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

```
##   train.pred
##     CH MM
## CH 441 44
## MM 77 238
```

```
(77+44)/(441+77+44+238)
```

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

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

```
##    test.pred
##      CH MM
## CH  151 17
## MM   33 69
```

```
(33+17)/(151+17+33+69)
```

```
## [1] 0.1851852
```

The radial basis kernel with default gamma creates 373 support vectors, out of which, 188 belong to level CHCH and remaining 185 belong to level MM. The classifier has a training error of 15.125% and a test error of 18.82% which is a slight improvement over linear kernel. We now use cross validation to find optimal gamma.

```
set.seed(755)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", 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
##     1
##
## - best performance: 0.165
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39375 0.04535738
## 2   0.01778279 0.39375 0.04535738
## 3   0.03162278 0.33750 0.08416254
```

```
## 4    0.05623413 0.19875 0.04543387
## 5    0.10000000 0.18875 0.04308019
## 6    0.17782794 0.18625 0.04693746
## 7    0.31622777 0.17375 0.05541823
## 8    0.56234133 0.16875 0.05597929
## 9    1.00000000 0.16500 0.05096295
## 10   1.77827941 0.17250 0.05296750
## 11   3.16227766 0.17875 0.05172376
## 12   5.62341325 0.17875 0.05272110
## 13  10.00000000 0.18750 0.05335937
```

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

```
##   train.pred
##     CH  MM
## CH 441  44
## MM  77 238
```

```
(77+44)/(441+77+44+238)
```

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

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

```
##    test.pred
##      CH  MM
## CH  151  17
## MM   33  69
```

```
(33+17)/(151+17+33+69)
```

```
## [1] 0.1851852
```

The result is exactly the same, which is better than linear kernel.

(g)

```
set.seed(8112)
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: 447
##
## ( 225 222 )
##
##
## 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 449  36
## MM 110 205
```

```
(110+36)/(449+36+110+205)
```

```
## [1] 0.1825
```

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

```
##     test.pred
##       CH  MM
## CH   153  15
## MM    45  57
```

```
(45+15)/(153+15+45+57)
```

```
## [1] 0.222222
```

Summary shows that polynomial kernel produces 447 support vectors, out of which, 225 belong to level CH and remaining 222 belong to level MM. This kernel produces a train error of 18.25% and a test error of 22.22% which are higher than the errors produces by radial kernel and by linear kernel.

```
set.seed(322)
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
## 3.162278
##
## - best performance: 0.18125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39500 0.04174992
## 2   0.01778279 0.37125 0.03387579
## 3   0.03162278 0.36625 0.03821086
## 4   0.05623413 0.33375 0.03775377
## 5   0.10000000 0.32125 0.04210189
## 6   0.17782794 0.23875 0.05604128
## 7   0.31622777 0.19625 0.04291869
## 8   0.56234133 0.20250 0.03374743
## 9   1.00000000 0.20125 0.03197764
## 10  1.77827941 0.19000 0.03622844
## 11  3.16227766 0.18125 0.03346329
## 12  5.62341325 0.18750 0.03726780
## 13 10.00000000 0.18875 0.03557562
```

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

```
##    train.pred
##      CH  MM
## CH 451  34
```

```
## MM  90 225
```

```
(34+90)/(451+34+90+225)
```

```
## [1] 0.155
```

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

```
##    test.pred
##      CH MM
## CH  153 14
## MM   41 61
```

```
(41+14)/(154+14+41+61)
```

```
## [1] 0.2037037
```

Tuning reduces the training error to 15.5% and test error to 20.37% which is worse than radial kernel and linear kernel.

(h) Overall, radial basis kernel seems to be producing minimum misclassification error on both train and test data.