Support Vector Machine

(using Caret package)

2019-September-19

Contents

HOMEWORK 1

Create a training set containing a random sample of 80% of the observations in the "juice.csv" data set using createDataPartition(). Create a test data set containing the remaining observations.Fit a SVM model to the training data using cost=0.01, with Purchase as the response and the other variables as predictors.Calculate the training and testing error for linear SVM model.Perform tune() function to find an optimal cost and re-run the SVM model to find training and testing error.Now perform the SVM model using "radial" and "polynomial" kernel in order to find which approach gives best result on this data

Data Summary:

The "juice.csv" data contains purchase information for Citrus Hill or Minute Maid orange juice. A description of the variables follows.

1. Purchase: A factor with levels CH and MM indicating whether the customer purchased Citrus Hill or Minute Maid Orange Juice

2. WeekofPurchase: Week of purchase

3. StoreID: Store ID

4. PriceCH: Price charged for CH

5. PriceMM: Price charged for MM

6. DiscCH: Discount offered for CH
7. DiscMM: Discount offered for MM
8. SpecialCH: Indicator of special on CH
9. SpecialMM: Indicator of special on MM
10. LoyalCH: Customer brand loyalty for CH
11. SalePriceMM: Sale price for MM
12. SalePriceCH: Sale price for CH
13. PriceDiff: Sale price of MM less sale price of CH
14. Store7: A factor with levels No and Yes indicating whether the sale is at Store 7
15. PctDiscMM: Percentage discount for MM
16. PctDiscCH: Percentage discount for CH
17. ListPriceDiff: List price of MM less list price of CH
18. STORE: Which of 5 possible stores the sale occured at

Question 1.

Create a training set containing a random sample of 80% of the observations in the "juice.csv" data set using createDataPartition(). Create a test data set containing the remaining observations.

```
#Read the juice.csv dataset
juice_df <- read.csv("juice.csv")
juice_df <- juice_df[,-c(4,5,14,17,18)]
juice_df$StoreID = as.factor(juice_df$StoreID)

dim(juice_df)
```

```
## [1] 1000   13
```

```
str(juice_df)
```

```
## 'data.frame':    1000 obs. of  13 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 2 1 2 1 1 2 1 1 1 1 ...
##  $ WeekofPurchase: int  237 258 242 271 276 240 248 270 266 274 ...
##  $ StoreID       : Factor w/ 5 levels "1","2","3","4",..: 2 5 3 2 2 1 3 1 2 5 ...
##  $ DiscCH        : num  0 0 0 0 0 0 0 0 0 0.47 ...
##  $ DiscMM        : num  0 0 0 0.06 0 0.3 0 0 0 0.54 ...
##  $ SpecialCH     : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ SpecialMM     : int  0 0 0 0 1 1 0 0 0 0 ...
##  $ LoyalCH       : num  0.4 0.90814 0.00721 0.78839 0.97251 ...
##  $ SalePriceMM   : num  1.99 2.18 2.23 2.12 2.18 1.69 2.23 2.18 2.18 1.59 ...
```

```
##  $ SalePriceCH   : num  1.75 1.86 1.99 1.86 1.99 1.75 1.99 1.86 1.86 1.39 ...
##  $ PriceDiff     : num  0.24 0.32 0.24 0.26 0.19 -0.06 0.24 0.32 0.32 0.2 ...
##  $ PctDiscMM     : num  0 0 0 0.0275 0 ...
##  $ PctDiscCH     : num  0 0 0 0 0 ...
```

Create data partition:

```r
#Step 1: Perform DataPartition
set.seed(123)
train_index <- createDataPartition(juice_df$Purchase, p=0.8, list=FALSE)
juice_train <- juice_df[train_index, ]
juice_test <- juice_df[-train_index,  ]
```

Question 2.

Fit a SVM model 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.

Linear SVM using C-Classification

```r
## Performance Evaluation ##
# generate predicted values based on training data
svm_linear <- svm(Purchase~., data=juice_train,kernel="linear",cost=0.01)
summary(svm_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  451
##
##  ( 225 226 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier creates 451 support vectors out of 800 training points. Out of these 225 belong to level CH and remaining 226 belong to level MM.

Question 3.

3. What are the training and test error rates?

```
pred_train_linear <- predict(svm_linear, juice_train)
conf.matrix<-(table(Predicted = pred_train_linear, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 434  85
##        MM  54 227
```

```
train_error_li<-1-(sum(diag(conf.matrix))) / sum(conf.matrix)
train_error_li
```

```
## [1] 0.174
```

```
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test_linear <- predict(svm_linear, juice_test)
conf.matrix<-(table(Predicted = pred_test_linear, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 111  20
##        MM  11  58
```

```
test_error_li<-1-(sum(diag(conf.matrix))) / sum(conf.matrix)
test_error_li
```

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

The Training Error for Linear SVM: 0.174
The Testing Error for Linear SVM: 0.155

Question 4.

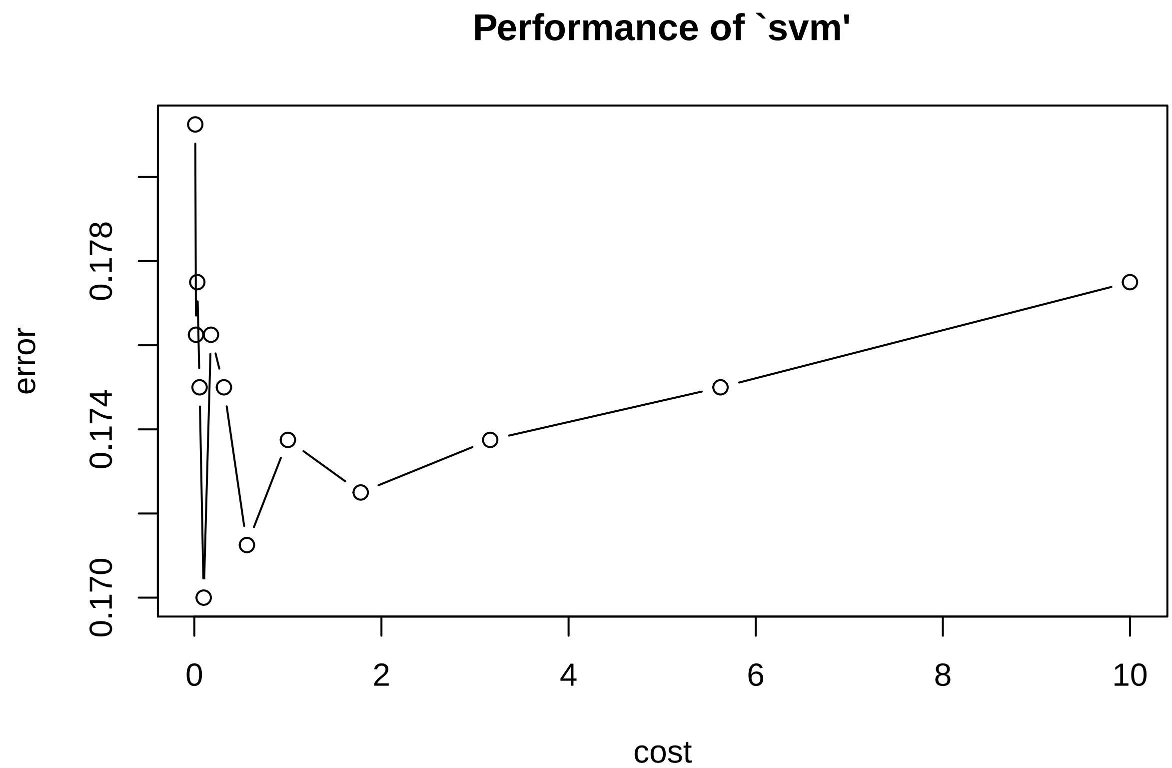
Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

Hyperparameter Optimization for Linear SVM

```
## Hyperparameter Optimization ##
set.seed(123)
tunesvm_linear <- tune(svm, Purchase~., data = juice_train,kernel="linear",
                  ranges = list(cost = 10^seq(-2,1, by = 0.25)))
summary(tunesvm_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.17
##
## - Detailed performance results:
##       cost error dispersion
## 1   0.0100 0.181     0.0409
## 2   0.0178 0.176     0.0388
## 3   0.0316 0.178     0.0343
## 4   0.0562 0.175     0.0323
## 5   0.1000 0.170     0.0296
## 6   0.1778 0.176     0.0309
## 7   0.3162 0.175     0.0306
## 8   0.5623 0.171     0.0264
## 9   1.0000 0.174     0.0285
## 10  1.7783 0.173     0.0287
## 11  3.1623 0.174     0.0224
## 12  5.6234 0.175     0.0212
## 13 10.0000 0.177     0.0227
```

```r
plot(tunesvm_linear)
```



THe optimal cost after tuning is 0.1

Question 5.

Compute and report the training and test error rates using this new value for cost.

```r
# Tuning shows that optimal cost is
svm1_linear <- svm(Purchase~., data=juice_train,kernel="linear",cost=tunesvm_linear$best.parameters$cos
summary(svm1_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "linear",
##     cost = tunesvm_linear$best.parameters$cost, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  449
##
##  ( 225 224 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## Performance Evaluation ##
# generate predicted values based on training data
pred_train1_linear <- predict(svm1_linear, juice_train)
conf.matrix<-(table(Predicted = pred_train1_linear, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 433  86
##        MM  55 226
```

```r
train_error_li_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
train_error_li_tune
```

```
## [1] 0.176
```

```r
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test1_linear <- predict(svm1_linear, juice_test)
conf.matrix<-(table(Predicted = pred_test1_linear, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 110  21
##        MM  12  57
```

```r
test_error_li_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
test_error_li_tune
```

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

The Training Error for Linear SVM after tuning: 0.176
The Testing Error for Linear SVM after tuning: 0.165

The training error decreases to 17.625% but test error slightly increases to 16.5% by using best cost.

Question 6.

Repeat parts (2.) through (5.) using a support vector machine with a radial kernel. Use the default value for gamma.

Radial SVM using C-Classification

```r
### Fit an SVM with radial kernel.
## SVM Model ## [CAN BE USED FOR CLASSIFICATION OR REGRESSION]

svm_radial <- svm(Purchase~., data=juice_train,kernel="radial",cost=0.01)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  627
##
##  ( 312 315 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## Performance Evaluation ##
# generate predicted values based on training data
pred_train_radial <- predict(svm_radial, juice_train)
conf.matrix<-(table(Predicted = pred_train_radial, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 488 312
##        MM   0   0
```

```r
train_error_radial<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
train_error_radial
```

```
## [1] 0.39
```

```r
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test_radial <- predict(svm_radial, juice_test)
conf.matrix<-(table(Predicted = pred_test_radial, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 122  78
##        MM   0   0
```

```r
test_error_radial<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
test_error_radial
```

```
## [1] 0.39
```

The Training Error for radial SVM: 0.39
The Testing Error for radial SVM: 0.39

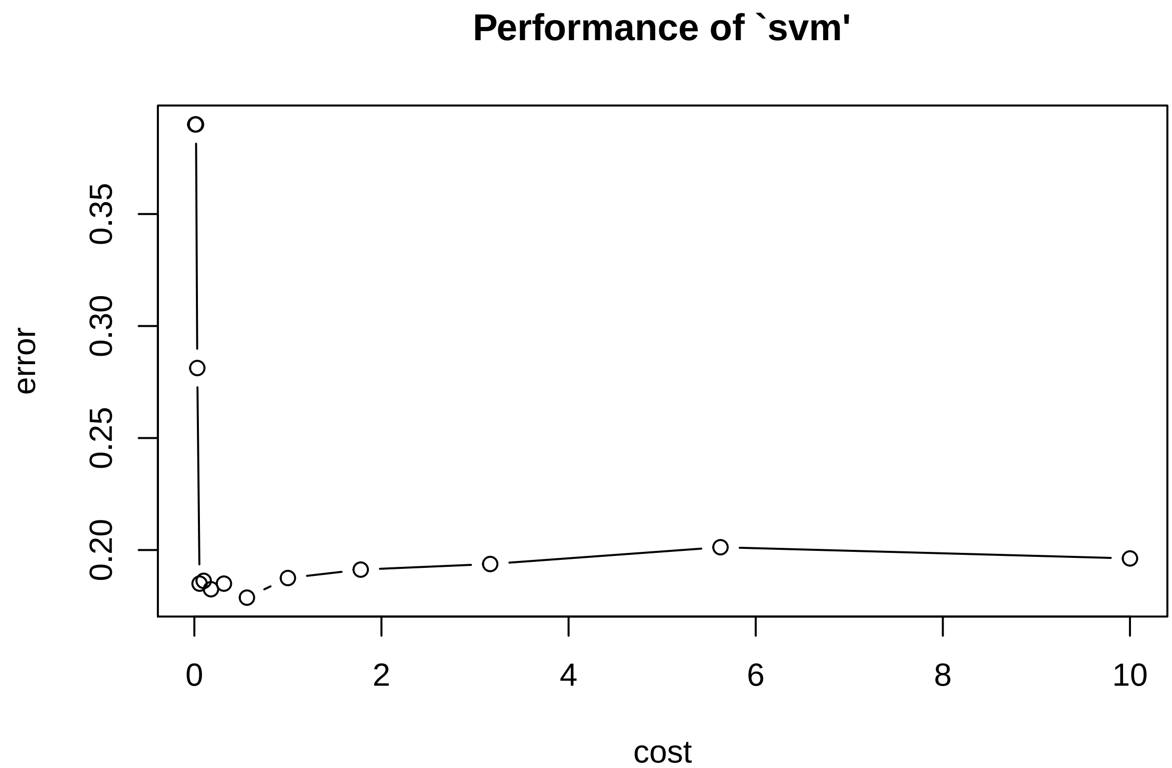
Hyperparameter Optimization for radial SVM

```r
## Hyperparameter Optimization ##
set.seed(123)
tunesvm_radial <- tune(svm, Purchase~., data = juice_train,kernel="radial",
                 ranges = list(cost = 10^seq(-2,1, by = 0.25)))

summary(tunesvm_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.562
##
## - best performance: 0.179
##
## - Detailed performance results:
##       cost error dispersion
## 1   0.0100 0.390     0.0642
## 2   0.0178 0.390     0.0642
## 3   0.0316 0.281     0.0732
## 4   0.0562 0.185     0.0436
## 5   0.1000 0.186     0.0410
## 6   0.1778 0.182     0.0355
## 7   0.3162 0.185     0.0376
## 8   0.5623 0.179     0.0349
## 9   1.0000 0.187     0.0429
## 10  1.7783 0.191     0.0373
## 11  3.1623 0.194     0.0388
## 12  5.6234 0.201     0.0375
## 13 10.0000 0.196     0.0460
```

```r
plot(tunesvm_radial)
```




```r
# Tuning shows that optimal cost is
svm1_radial <- svm(Purchase~., data=juice_train,kernel="radial",cost=tunesvm_radial$best.parameters$cost)
summary(svm1_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "radial",
##     cost = tunesvm_radial$best.parameters$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.562
##
## Number of Support Vectors:  399
##
##  ( 200 199 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## Performance Evaluation ##
# generate predicted values based on training data
pred_train1_radial <- predict(svm1_radial, juice_train)
conf.matrix<-(table(Predicted = pred_train1_radial, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 442  81
##        MM  46 231
```

```r
train_error_radial_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
train_error_radial_tune
```

```
## [1] 0.159
```

```r
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test1_radial1 <- predict(svm1_radial, juice_test)
conf.matrix<-(table(Predicted = pred_test1_radial1, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 113  24
##        MM   9  54
```

```r
test_error_radial_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
test_error_radial_tune
```

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

The Training Error for radial SVM after tuning: 0.176
The Testing Error for radial SVM after tuning: 0.165

Tuning slightly decreases training error to 17.625% and slightly increases test error to 16.5% which is still better than linear kernel.

Question 7.

Repeat parts (2.) through (5.) using a support vector machine with a polynomial kernel. Set degree=2.

Polynomial SVM using C-Classification

```r
### Fit an SVM with polynomial kernel.
## SVM Model ## [CAN BE USED FOR CLASSIFICATION OR REGRESSION]

svm_polynomial <- svm(Purchase~., data=juice_train,kernel="polynomial",cost=0.01,degree=2)
summary(svm_polynomial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "polynomial",
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  628
##
##  ( 312 316 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
## Performance Evaluation ##
# generate predicted values based on training data
pred_train_poly <- predict(svm_polynomial, juice_train)
conf.matrix<-(table(Predicted = pred_train_poly, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 488 312
##        MM   0   0
```

```
train_error_poly<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
train_error_poly
```

```
## [1] 0.39
```

```
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test_poly <- predict(svm_polynomial, juice_test)
conf.matrix<-(table(Predicted = pred_test_poly, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 122  78
##        MM   0   0
```

```r
test_error_poly<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
test_error_poly
```

```
## [1] 0.39
```

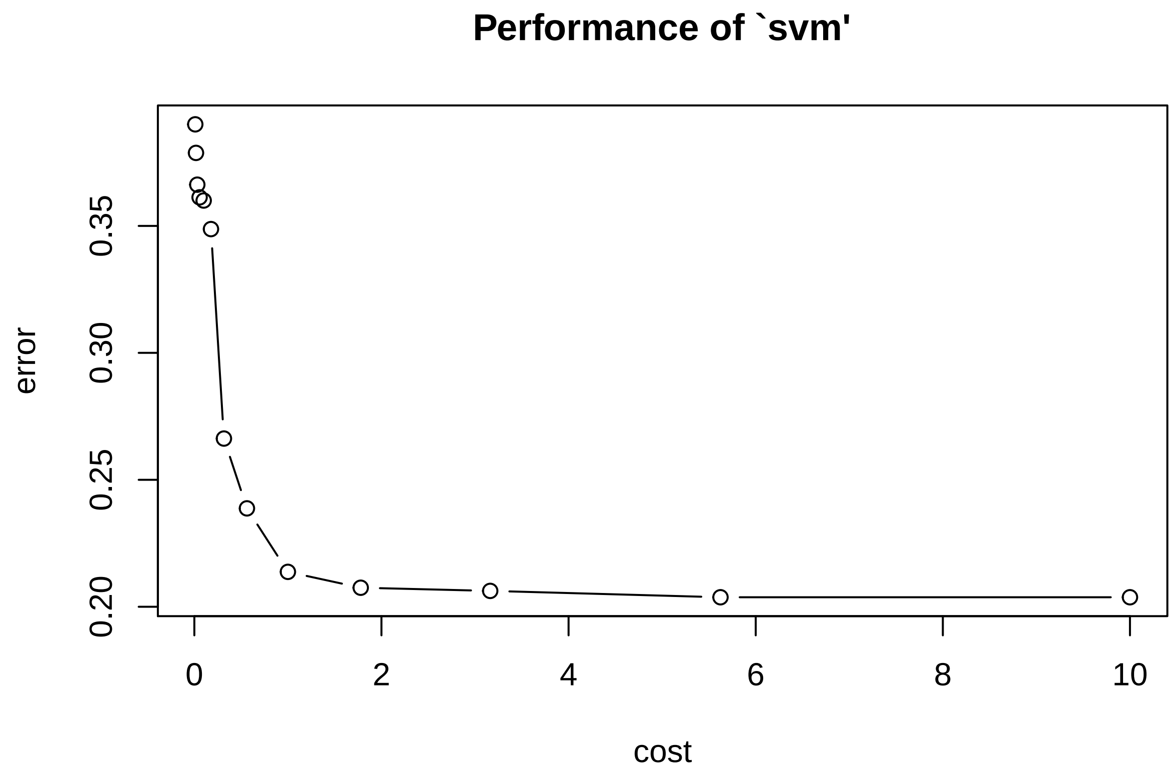
The Training Error for polynomial SVM: 0.39
The Testing Error for polynomial SVM: 0.39

Hyperparameter Optimization for polynomial SVM

```r
## Hyperparameter Optimization ##
set.seed(123)
tunesvm_ploy <- tune(svm, Purchase~., data = juice_train,kernel="polynomial",degree=2,
                       ranges = list(cost = 10^seq(-2,1, by = 0.25)))
summary(tunesvm_ploy)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  5.62
##
## - best performance: 0.204
##
## - Detailed performance results:
##       cost error dispersion
## 1   0.0100 0.390     0.0642
## 2   0.0178 0.379     0.0719
## 3   0.0316 0.366     0.0615
## 4   0.0562 0.361     0.0557
## 5   0.1000 0.360     0.0565
## 6   0.1778 0.349     0.0548
## 7   0.3162 0.266     0.0490
## 8   0.5623 0.239     0.0443
## 9   1.0000 0.214     0.0551
## 10  1.7783 0.208     0.0578
## 11  3.1623 0.206     0.0525
## 12  5.6234 0.204     0.0521
## 13 10.0000 0.204     0.0497
```

```r
plot(tunesvm_ploy)
```




```
# Tuning shows that optimal cost is
svm1_ploy <- svm(Purchase~., data=juice_train,kernel="polynomial",cost=tunesvm_ploy$best.parameters$cost)
summary(svm1_ploy)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = juice_train, kernel = "polynomial",
##     cost = tunesvm_ploy$best.parameters$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5.62
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  386
##
##  ( 191 195 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## Performance Evaluation ##
# generate predicted values based on training data
pred_train1_ploy <- predict(svm1_ploy, juice_train)
conf.matrix<-(table(Predicted = pred_train1_ploy, Actual = juice_train$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 452  97
##        MM  36 215
```

```r
train_error_poly_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
train_error_poly_tune
```

```
## [1] 0.166
```

```r
## Performance Evaluation ##
# generate predicted values based on Testing data
pred_test1_poly <- predict(svm1_ploy, juice_test)
conf.matrix<-(table(Predicted = pred_test1_poly, Actual = juice_test$Purchase))
conf.matrix
```

```
##          Actual
## Predicted  CH  MM
##        CH 112  31
##        MM  10  47
```

```r
test_error_poly_tune<-1-(sum(diag(conf.matrix)) / sum(conf.matrix))
test_error_poly_tune
```

```
## [1] 0.205
```

The Training Error for polynomial SVM after tuning : 0.166
The Testing Error for polynomial SVM after tuning: 0.205

Tuning reduces the training error to 16.625% and test error to 20.5% which is worse than radial kernel but slightly better than linear kernel.

Question 8.

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

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