Ghoshal_Gourav_HW6

Gourav Ghoshal

November 28, 2018

Problem 1

In this problem, you will use support vector approaches to predict whether a given car gets high or low gas mileage based on the Auto data set in the ISLR package.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median. Use this variable as response in the following analysis.

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.4.2
```

```r
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                        name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

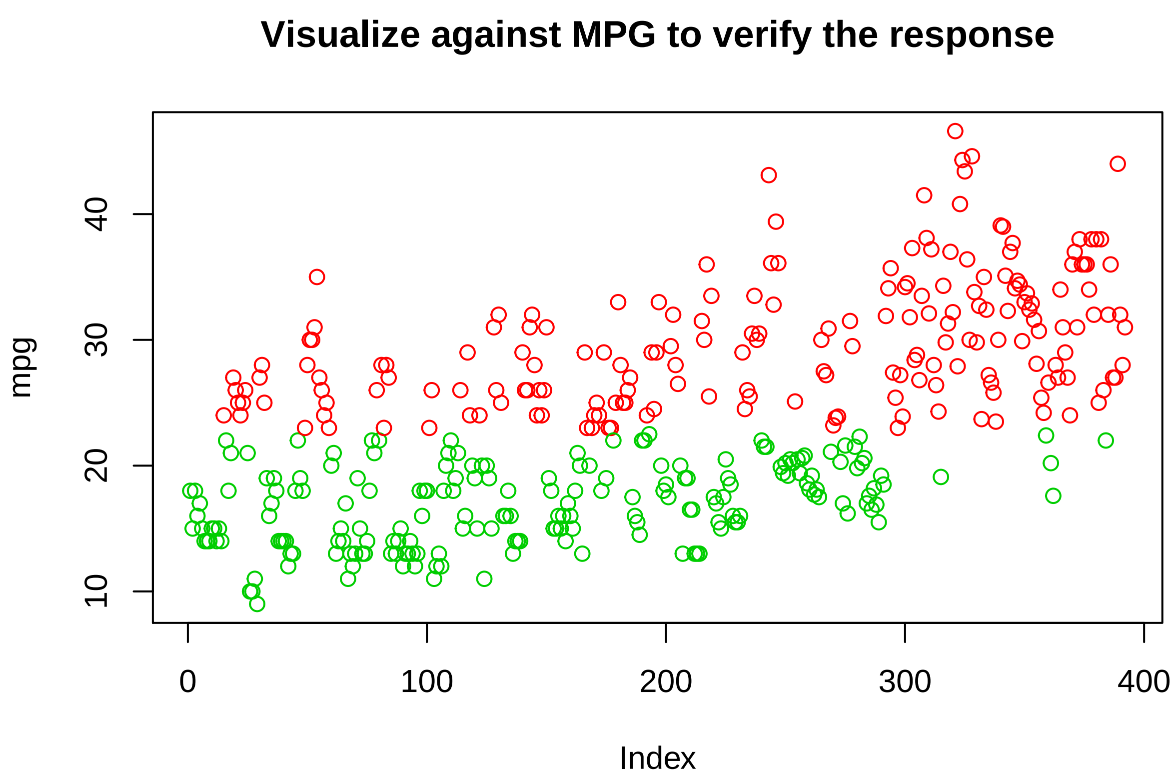
```r
data = Auto
data$response = ifelse(data$mpg >= median(data$mpg), 1, 0)
tail(data)
```

```
##     mpg cylinders displacement horsepower weight acceleration year origin
## 392  27         4          151         90   2950         17.3   82      1
## 393  27         4          140         86   2790         15.6   82      1
## 394  44         4           97         52   2130         24.6   82      2
## 395  32         4          135         84   2295         11.6   82      1
## 396  28         4          120         79   2625         18.6   82      1
## 397  31         4          119         82   2720         19.4   82      1
##                  name response
## 392 chevrolet camaro        1
## 393  ford mustang gl        1
## 394        vw pickup        1
## 395    dodge rampage        1
## 396      ford ranger        1
```

```
## 397        chevy s-10        1
```

(b) Fit a support vector classifier to the data with various values of cost, to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

```r
# Let's plot the data
attach(data)
plot(mpg, col = (3-response), main = 'Visualize against MPG to verify the response')
```




```r
# fitting SVM
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.2
```

```r
data_new = data.frame(x = data[,-c(response,mpg)], y = as.factor(data$response))

set.seed(42)
tune.out=tune(svm ,y~.,data=data_new, kernel ="linear",
              ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.08435897
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.13012821 0.05596578
## 2 1e-02 0.08923077 0.04999616
## 3 1e-01 0.09935897 0.05820444
## 4 1e+00 0.08935897 0.05024613
## 5 5e+00 0.08435897 0.05287086
## 6 1e+01 0.08435897 0.05287086
## 7 1e+02 0.08692308 0.05707025
```

```r
# C= 5 and C = 10, both gave the minimum CV error
# Since higher value of C (cost) gives narrower margin, I decide to select it. cost = 10

## prediction
model = tune.out$best.model
y_hat = predict(model, data_new)

table(predict =y_hat, truth = data_new$y)
```

```
##        truth
## predict   0   1
##       0 174  11
##       1  22 185
```

(c) Now repeat (b), this time using SVMs with radial and polynomial kernels, with different values of gamma, degree and cost. Comment on your results.

```r
set.seed (1)

tune.out.radial=tune(svm , y~., data=data_new, kernel ="radial",
              ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),
                           degree = c(1,2,3,4,5),
                           gamma=c(0.5,1,2,3,4)))
summary(tune.out.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree gamma
##     1      1     1
##
## - best performance: 0.07391026
##
## - Detailed performance results:
```

```
##       cost degree gamma      error dispersion
## 1   1e-01      1   0.5 0.08923077 0.05559893
## 2   1e+00      1   0.5 0.07391026 0.05315495
## 3   1e+01      1   0.5 0.07897436 0.04567335
## 4   1e+02      1   0.5 0.10448718 0.06862934
## 5   1e+03      1   0.5 0.10448718 0.06072287
## 6   1e-01      2   0.5 0.08923077 0.05559893
## 7   1e+00      2   0.5 0.07391026 0.05315495
## 8   1e+01      2   0.5 0.07897436 0.04567335
## 9   1e+02      2   0.5 0.10448718 0.06862934
## 10  1e+03      2   0.5 0.10448718 0.06072287
## 11  1e-01      3   0.5 0.08923077 0.05559893
## 12  1e+00      3   0.5 0.07391026 0.05315495
## 13  1e+01      3   0.5 0.07897436 0.04567335
## 14  1e+02      3   0.5 0.10448718 0.06862934
## 15  1e+03      3   0.5 0.10448718 0.06072287
## 16  1e-01      4   0.5 0.08923077 0.05559893
## 17  1e+00      4   0.5 0.07391026 0.05315495
## 18  1e+01      4   0.5 0.07897436 0.04567335
## 19  1e+02      4   0.5 0.10448718 0.06862934
## 20  1e+03      4   0.5 0.10448718 0.06072287
## 21  1e-01      5   0.5 0.08923077 0.05559893
## 22  1e+00      5   0.5 0.07391026 0.05315495
## 23  1e+01      5   0.5 0.07897436 0.04567335
## 24  1e+02      5   0.5 0.10448718 0.06862934
## 25  1e+03      5   0.5 0.10448718 0.06072287
## 26  1e-01      1   1.0 0.08923077 0.06408390
## 27  1e+00      1   1.0 0.07391026 0.05176240
## 28  1e+01      1   1.0 0.08916667 0.05799733
## 29  1e+02      1   1.0 0.10698718 0.07386765
## 30  1e+03      1   1.0 0.10698718 0.07386765
## 31  1e-01      2   1.0 0.08923077 0.06408390
## 32  1e+00      2   1.0 0.07391026 0.05176240
## 33  1e+01      2   1.0 0.08916667 0.05799733
## 34  1e+02      2   1.0 0.10698718 0.07386765
## 35  1e+03      2   1.0 0.10698718 0.07386765
## 36  1e-01      3   1.0 0.08923077 0.06408390
## 37  1e+00      3   1.0 0.07391026 0.05176240
## 38  1e+01      3   1.0 0.08916667 0.05799733
## 39  1e+02      3   1.0 0.10698718 0.07386765
## 40  1e+03      3   1.0 0.10698718 0.07386765
## 41  1e-01      4   1.0 0.08923077 0.06408390
## 42  1e+00      4   1.0 0.07391026 0.05176240
## 43  1e+01      4   1.0 0.08916667 0.05799733
## 44  1e+02      4   1.0 0.10698718 0.07386765
## 45  1e+03      4   1.0 0.10698718 0.07386765
## 46  1e-01      5   1.0 0.08923077 0.06408390
## 47  1e+00      5   1.0 0.07391026 0.05176240
## 48  1e+01      5   1.0 0.08916667 0.05799733
## 49  1e+02      5   1.0 0.10698718 0.07386765
## 50  1e+03      5   1.0 0.10698718 0.07386765
## 51  1e-01      1   2.0 0.13512821 0.05778941
## 52  1e+00      1   2.0 0.07634615 0.05879482
## 53  1e+01      1   2.0 0.09423077 0.07220895
```

```
## 54  1e+02      1   2.0 0.09923077 0.07799865
## 55  1e+03      1   2.0 0.09923077 0.07799865
## 56  1e-01      2   2.0 0.13512821 0.05778941
## 57  1e+00      2   2.0 0.07634615 0.05879482
## 58  1e+01      2   2.0 0.09423077 0.07220895
## 59  1e+02      2   2.0 0.09923077 0.07799865
## 60  1e+03      2   2.0 0.09923077 0.07799865
## 61  1e-01      3   2.0 0.13512821 0.05778941
## 62  1e+00      3   2.0 0.07634615 0.05879482
## 63  1e+01      3   2.0 0.09423077 0.07220895
## 64  1e+02      3   2.0 0.09923077 0.07799865
## 65  1e+03      3   2.0 0.09923077 0.07799865
## 66  1e-01      4   2.0 0.13512821 0.05778941
## 67  1e+00      4   2.0 0.07634615 0.05879482
## 68  1e+01      4   2.0 0.09423077 0.07220895
## 69  1e+02      4   2.0 0.09923077 0.07799865
## 70  1e+03      4   2.0 0.09923077 0.07799865
## 71  1e-01      5   2.0 0.13512821 0.05778941
## 72  1e+00      5   2.0 0.07634615 0.05879482
## 73  1e+01      5   2.0 0.09423077 0.07220895
## 74  1e+02      5   2.0 0.09923077 0.07799865
## 75  1e+03      5   2.0 0.09923077 0.07799865
## 76  1e-01      1   3.0 0.30128205 0.11716574
## 77  1e+00      1   3.0 0.07634615 0.05867200
## 78  1e+01      1   3.0 0.09673077 0.06968463
## 79  1e+02      1   3.0 0.09923077 0.07316609
## 80  1e+03      1   3.0 0.09923077 0.07316609
## 81  1e-01      2   3.0 0.30128205 0.11716574
## 82  1e+00      2   3.0 0.07634615 0.05867200
## 83  1e+01      2   3.0 0.09673077 0.06968463
## 84  1e+02      2   3.0 0.09923077 0.07316609
## 85  1e+03      2   3.0 0.09923077 0.07316609
## 86  1e-01      3   3.0 0.30128205 0.11716574
## 87  1e+00      3   3.0 0.07634615 0.05867200
## 88  1e+01      3   3.0 0.09673077 0.06968463
## 89  1e+02      3   3.0 0.09923077 0.07316609
## 90  1e+03      3   3.0 0.09923077 0.07316609
## 91  1e-01      4   3.0 0.30128205 0.11716574
## 92  1e+00      4   3.0 0.07634615 0.05867200
## 93  1e+01      4   3.0 0.09673077 0.06968463
## 94  1e+02      4   3.0 0.09923077 0.07316609
## 95  1e+03      4   3.0 0.09923077 0.07316609
## 96  1e-01      5   3.0 0.30128205 0.11716574
## 97  1e+00      5   3.0 0.07634615 0.05867200
## 98  1e+01      5   3.0 0.09673077 0.06968463
## 99  1e+02      5   3.0 0.09923077 0.07316609
## 100 1e+03      5   3.0 0.09923077 0.07316609
## 101 1e-01      1   4.0 0.53820513 0.07278281
## 102 1e+00      1   4.0 0.08660256 0.05385501
## 103 1e+01      1   4.0 0.10179487 0.06661035
## 104 1e+02      1   4.0 0.10179487 0.06966782
## 105 1e+03      1   4.0 0.10179487 0.06966782
## 106 1e-01      2   4.0 0.53820513 0.07278281
## 107 1e+00      2   4.0 0.08660256 0.05385501
```

```
## 108 1e+01      2   4.0 0.10179487 0.06661035
## 109 1e+02      2   4.0 0.10179487 0.06966782
## 110 1e+03      2   4.0 0.10179487 0.06966782
## 111 1e-01      3   4.0 0.53820513 0.07278281
## 112 1e+00      3   4.0 0.08660256 0.05385501
## 113 1e+01      3   4.0 0.10179487 0.06661035
## 114 1e+02      3   4.0 0.10179487 0.06966782
## 115 1e+03      3   4.0 0.10179487 0.06966782
## 116 1e-01      4   4.0 0.53820513 0.07278281
## 117 1e+00      4   4.0 0.08660256 0.05385501
## 118 1e+01      4   4.0 0.10179487 0.06661035
## 119 1e+02      4   4.0 0.10179487 0.06966782
## 120 1e+03      4   4.0 0.10179487 0.06966782
## 121 1e-01      5   4.0 0.53820513 0.07278281
## 122 1e+00      5   4.0 0.08660256 0.05385501
## 123 1e+01      5   4.0 0.10179487 0.06661035
## 124 1e+02      5   4.0 0.10179487 0.06966782
## 125 1e+03      5   4.0 0.10179487 0.06966782
```

```
## Minimum error occur @ cost = 1,degree = 2, gamma = 1

set.seed (1)

tune.out.poly=tune(svm , y~., data=data_new, kernel ="polynomial",
              ranges =list(cost=c(0.1 ,1 ,10 ,100),
                           degree = c(1,2,3,4,5),
                           gamma=c(0.5,1)))
summary(tune.out.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree gamma
##     1      3   0.5
##
## - best performance: 0.07397436
##
## - Detailed performance results:
##     cost degree gamma      error dispersion
## 1    0.1      1   0.5 0.09185897 0.06069132
## 2    1.0      1   0.5 0.09435897 0.06395767
## 3   10.0      1   0.5 0.08935897 0.05443327
## 4  100.0      1   0.5 0.08679487 0.05567394
## 5    0.1      2   0.5 0.21923077 0.09233341
## 6    1.0      2   0.5 0.19615385 0.07426805
## 7   10.0      2   0.5 0.17083333 0.07329107
## 8  100.0      2   0.5 0.16820513 0.07983348
## 9    0.1      3   0.5 0.07653846 0.05269354
## 10   1.0      3   0.5 0.07397436 0.05464674
## 11  10.0      3   0.5 0.09179487 0.05146406
## 12 100.0      3   0.5 0.08929487 0.04398809
## 13   0.1      4   0.5 0.16064103 0.09334022
```

```
## 14   1.0      4   0.5 0.14525641 0.07506797
## 15  10.0      4   0.5 0.16064103 0.07230196
## 16 100.0      4   0.5 0.18115385 0.08023736
## 17   0.1      5   0.5 0.10461538 0.06888486
## 18   1.0      5   0.5 0.09173077 0.03417795
## 19  10.0      5   0.5 0.10705128 0.05346238
## 20 100.0      5   0.5 0.11474359 0.04855649
## 21   0.1      1   1.0 0.09948718 0.06331482
## 22   1.0      1   1.0 0.09192308 0.05827672
## 23  10.0      1   1.0 0.08935897 0.05443327
## 24 100.0      1   1.0 0.08679487 0.05567394
## 25   0.1      2   1.0 0.18846154 0.07426805
## 26   1.0      2   1.0 0.17576923 0.07691940
## 27  10.0      2   1.0 0.16570513 0.07916097
## 28 100.0      2   1.0 0.17083333 0.08345228
## 29   0.1      3   1.0 0.07397436 0.05464674
## 30   1.0      3   1.0 0.09429487 0.05250366
## 31  10.0      3   1.0 0.09192308 0.04407224
## 32 100.0      3   1.0 0.10724359 0.04971433
## 33   0.1      4   1.0 0.14782051 0.07866510
## 34   1.0      4   1.0 0.16314103 0.07322999
## 35  10.0      4   1.0 0.16339744 0.07437153
## 36 100.0      4   1.0 0.17878205 0.06764193
## 37   0.1      5   1.0 0.09423077 0.05370011
## 38   1.0      5   1.0 0.10692308 0.05300334
## 39  10.0      5   1.0 0.12237179 0.05214415
## 40 100.0      5   1.0 0.12237179 0.05214415
```

```
## Minimum error occur @ cost = 0.1,degree = 3, gamma = 1
```

Problem 2

This problem uses the OJ data set in the ISLR package

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

```r
library(ISLR)
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
```

```
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

```r
summary(OJ)
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH         PriceMM     
##  CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690  
##  MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990  
##           Median :257.0   Median :3.00   Median :1.860   Median :2.090  
##           Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085  
##           3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180  
##           Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290  
##      DiscCH            DiscMM         SpecialCH        SpecialMM     
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000  
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617  
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000  
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000  
##     LoyalCH          SalePriceMM     SalePriceCH      PriceDiff      
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700  
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000  
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300  
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465  
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200  
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400  
##  Store7      PctDiscMM        PctDiscCH       ListPriceDiff  
##  No :714   Min.   :0.0000   Min.   :0.00000   Min.   :0.000  
##  Yes:356   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140  
##            Median :0.0000   Median :0.00000   Median :0.240  
##            Mean   :0.0593   Mean   :0.02731   Mean   :0.218  
##            3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300  
##            Max.   :0.4020   Max.   :0.25269   Max.   :0.440  
##      STORE      
##  Min.   :0.000  
##  1st Qu.:0.000  
##  Median :2.000  
##  Mean   :1.631  
##  3rd Qu.:3.000  
##  Max.   :4.000  
```

```r
## Creating train-test
set.seed(101)
train_index = sample.int(dim(OJ), size = 800)
train_data = OJ[train_index,]
test_data = OJ[-train_index,]
```

(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_fit = svm(train_data$Purchase ~., data = train_data,
              kernel = 'linear', cost = 0.01)

summary(svm_fit)
```

```
##
## Call:
## svm(formula = train_data$Purchase ~ ., data = train_data, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  433
##
##  ( 218 215 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
## There are 433 support vectors, 218 from CH and 215 from MM
```

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

```
y_hat_train = predict(svm_fit, train_data)
table(predict =y_hat_train, truth = train_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 430  77
##      MM  57 236
```

```
accuracy_train = (430+236)/(430+77+57+236)
accuracy_train
```

```
## [1] 0.8325
```

```
y_hat_test = predict(svm_fit, test_data)
table(predict =y_hat_test, truth = test_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 148  27
```

```
##      MM  18  77
```

```
accuracy_test = (148+77)/(148+27+18+77)
accuracy_test
```

```
## [1] 0.8333333
```

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

```
tune.out=tune(svm ,y~.,data=data_new, kernel ="linear",
              ranges =list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))
summary(tune.out) ## optimal cost = 0.5
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.07923077
##
## - Detailed performance results:
##    cost      error dispersion
## 1  0.01 0.08935897 0.03040982
## 2  0.05 0.09705128 0.02682318
## 3  0.10 0.09448718 0.02465186
## 4  0.50 0.07923077 0.03737692
## 5  1.00 0.08179487 0.03180130
## 6  5.00 0.08685897 0.03483004
## 7 10.00 0.08685897 0.03483004
```

```
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = data_new, ranges = list(cost = c(0.01,
##     0.05, 0.1, 0.5, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.5
##       gamma:  0.1428571
##
## Number of Support Vectors:  89
```

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

```
svm_fit_opt = svm(train_data$Purchase ~., data = train_data,
              kernel = 'linear', cost = 0.5)
```

```r
y_hat_train_opt = predict(svm_fit_opt, train_data)
table(predict =y_hat_train_opt, truth = train_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 429  77
##      MM  58 236
```

```r
accuracy_train_opt = (429+236)/(429+77+58+236)
accuracy_train_opt
```

```
## [1] 0.83125
```

```r
y_hat_test_opt = predict(svm_fit, test_data)
table(predict =y_hat_test_opt, truth = test_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 148  27
##      MM  18  77
```

```r
accuracy_test = (148+77)/(148+27+18+77)
accuracy_test
```

```
## [1] 0.8333333
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the tune() function to select an optimal cost and gamma.

```r
# with cost = 0.01
svm_fit = svm(train_data$Purchase ~., data = train_data,
              kernel = 'radial', cost = 0.01)

summary(svm_fit)
```

```
##
## Call:
## svm(formula = train_data$Purchase ~ ., data = train_data, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  628
##
##  ( 315 313 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
## There are 628 support vectors, 315 from CH and 313 from MM class
# Radial Kernel
# Errors using cost = 0.01
y_hat_train = predict(svm_fit, train_data)
table(predict =y_hat_train, truth = train_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 487 313
##      MM   0   0
```

```
accuracy_train = (487)/(487+0+313+0)
accuracy_train
```

```
## [1] 0.60875
```

```
y_hat_test = predict(svm_fit, test_data)
table(predict =y_hat_test, truth = test_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 166 104
##      MM   0   0
```

```
accuracy_test = (166)/(166+0+104+0)
accuracy_test
```

```
## [1] 0.6148148
```

```
## cv -  to find opt params
new_data = data.frame(x = train_data[,-1], y = as.factor(train_data$Purchase))

svm_tune=tune(svm , y~., data=new_data,
              kernel='radial',ranges =list(
                cost=c(0.01 ,0.1 ,1 ,10),gamma=c(0.1,0.5,1,2,3,4)))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.1
##
## - best performance: 0.17375
##
## - Detailed performance results:
##     cost gamma   error dispersion
## 1   0.01   0.1 0.39125 0.06153420
## 2   0.10   0.1 0.18500 0.04706674
## 3   1.00   0.1 0.17375 0.04839436
## 4  10.00   0.1 0.19375 0.04340139
## 5   0.01   0.5 0.39125 0.06153420
## 6   0.10   0.5 0.28500 0.04116363
```

```
## 7   1.00   0.5 0.20000 0.04639804
## 8  10.00   0.5 0.22250 0.05230785
## 9   0.01   1.0 0.39125 0.06153420
## 10  0.10   1.0 0.34375 0.06380580
## 11  1.00   1.0 0.21125 0.05635022
## 12 10.00   1.0 0.22625 0.05382908
## 13  0.01   2.0 0.39125 0.06153420
## 14  0.10   2.0 0.37250 0.06476453
## 15  1.00   2.0 0.22250 0.04594683
## 16 10.00   2.0 0.24875 0.06022239
## 17  0.01   3.0 0.39125 0.06153420
## 18  0.10   3.0 0.38750 0.06152010
## 19  1.00   3.0 0.24000 0.04816061
## 20 10.00   3.0 0.25125 0.05478810
## 21  0.01   4.0 0.39125 0.06153420
## 22  0.10   4.0 0.39125 0.06153420
## 23  1.00   4.0 0.23500 0.05096295
## 24 10.00   4.0 0.25750 0.05374838
```

```r
# optimal cost = 1, optimal gamma = 0.1

## best radial kernel performance -
best_radial = svm(y~., data=new_data, kernel='radial',
                  cost = 1, gamma = 0.1)
y_hat_train = predict(best_radial, new_data)
table(predict =y_hat_train, truth = new_data$y)
```

```
##        truth
## predict  CH  MM
##      CH 447  72
##      MM  40 241
```

```r
accuracy_train = (447+241)/(447+40+72+241)
accuracy_train # 0.86
```

```
## [1] 0.86
```

```r
new_test_data = data.frame(x = test_data[,-1], y = as.factor(test_data$Purchase))

y_hat_test = predict(best_radial, new_test_data)
table(predict =y_hat_test, truth = new_test_data$y)
```

```
##        truth
## predict  CH  MM
##      CH 148  30
##      MM  18  74
```

```r
accuracy_test = (148+74)/(148+18+30+74)
accuracy_test # 0.822
```

```
## [1] 0.8222222
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2. Use the tune() function to select an optimal cost.

```r
# Polynomial kernel
# with cost = 0.01
svm_fit = svm(train_data$Purchase ~., data = train_data,
              kernel = 'polynomial', degree = 2)

summary(svm_fit)
```

```
##
## Call:
## svm(formula = train_data$Purchase ~ ., data = train_data, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  451
##
##  ( 227 224 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## There are 451 support vectors, 227 from CH and 224 from MM class

# Errors using degree = 2
y_hat_train = predict(svm_fit, train_data)
table(predict =y_hat_train, truth = train_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 459 112
##      MM  28 201
```

```r
accuracy_train = (459+201)/(459+112+28+201)
accuracy_train
```

```
## [1] 0.825
```

```r
y_hat_test = predict(svm_fit, test_data)
table(predict =y_hat_test, truth = test_data$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 153  41
##      MM  13  63
```

```r
accuracy_test = (153+63)/(153+41+13+63)
accuracy_test
```

```
## [1] 0.8
```

```r
new_data = data.frame(x = train_data[,-1], y = as.factor(train_data$Purchase))

svm_tune=tune(svm , y~., data=new_data,
              kernel="polynomial",ranges =list(cost=c(0.01, 0.1 ,1 ,10
                                                      ,100, 1000),
                                               degree = 2))
summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      2
##
## - best performance: 0.17125
##
## - Detailed performance results:
##    cost degree   error dispersion
## 1 1e-02      2 0.39000 0.03670453
## 2 1e-01      2 0.32000 0.05688683
## 3 1e+00      2 0.19375 0.06802012
## 4 1e+01      2 0.18500 0.04362084
## 5 1e+02      2 0.17375 0.04185375
## 6 1e+03      2 0.17125 0.03230175
```

```r
# optimal cost = 100

## best polynomial kernel performance -
best_poly = svm(y~., data=new_data, kernel='polynomial', degree = 2,
                cost = 100)
y_hat_train = predict(best_poly, new_data)
table(predict =y_hat_train, truth = new_data$y)
```

```
##        truth
## predict  CH  MM
##      CH 442  57
##      MM  45 256
```

```r
accuracy_train = (442+256)/(442+45+57+256)
accuracy_train # 0.8725
```

```
## [1] 0.8725
```

```r
new_test_data = data.frame(x = test_data[,-1], y = as.factor(test_data$Purchase))

y_hat_test = predict(best_poly, new_test_data)
table(predict =y_hat_test, truth = new_test_data$y)
```

```
##        truth
```

```
## predict  CH  MM
##      CH 147  26
##      MM  19  78
```

```r
accuracy_test = (147+78)/(147+19+26+78)
accuracy_test # 0.833
```

```
## [1] 0.8333333
```

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

```
## Overall the Polynomial kernel performs better as both the training and testing accuracy is higher
```