SDM_Assignment5_3

Sri Balaji Muruganandam

9/12/2021

Setting Working Directory

```
rm(list = ls())
setwd("G:\\SDM_Sem01\\Assignment5")
```

Importing necessary libraries

```
options(warn=-1)
require(ISLR2)
```

```
## Loading required package: ISLR2
```

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
require(e1071)
```

```
## Loading required package: e1071
```

```
require(MASS)
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
options(warn=0)
```

Loading the dataset

```
data("OJ")
dim(OJ)
```

```
## [1] 1070   18
```

Exploring the high level overview of the data

```
names(OJ)
```

```
##  [1] "Purchase"       "WeekofPurchase" "StoreID"        "PriceCH"       
##  [5] "PriceMM"        "DiscCH"         "DiscMM"         "SpecialCH"     
##  [9] "SpecialMM"      "LoyalCH"        "SalePriceMM"    "SalePriceCH"   
## [13] "PriceDiff"      "Store7"         "PctDiscMM"      "PctDiscCH"     
## [17] "ListPriceDiff"  "STORE"
```

```
head(OJ,5)
```

```
##   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
##   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
##   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
```

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

Splitting the data into training and the test data

```
set.seed(23)
random_index = sample(c(1:nrow(OJ)), size = round(8/10 * nrow(OJ)), replace = FALSE)
train_data = OJ[random_index,]
test_data = OJ[-random_index,]

y_train_data = as.numeric(OJ$Purchase)-1
y_test_data = as.numeric(OJ$Purchase)-1
dim(train_data)
```

```
## [1] 856  18
```

```
dim(test_data)
```

```
## [1] 214  18
```

(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.

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'linear', cost = 0.01)
```

Getting the summary of the model

```
summary(model_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  461
##
##  ( 229 232 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"           
##  [5] "degree"          "gamma"           "coef0"           "nu"             
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"        
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"        
## [17] "nSV"             "labels"          "SV"              "index"          
## [21] "rho"             "compprob"        "probA"           "probB"          
## [25] "sigma"           "coefs"           "na.action"       "fitted"         
## [29] "decision.values" "terms"
```

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

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.5158879
```

Predicting for the new set of values - Test Data

```
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.5785047
```

The training error is 0.5158879 and the test error is 0.5785047

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

```
model_tuning <- tune(svm, Purchase~., data = train_data, kernel = "linear", ranges = list(0.0
1, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10))
```

Summary of tuned model

```
summary(model_tuning)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.1612449
```

```
names(model_tuning)
```

```
## [1] "best.parameters"  "best.performance" "method"           "nparcomb"
## [5] "train.ind"        "sampling"         "performances"     "best.model"
```

Best value for cost

```
print(model_tuning$best.parameters)
```

```
##   Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
## 1 0.01  0.1  0.5    1  3.5  5.5  7.5   10
```

```
print(model_tuning$best.performance)
```

```
## [1] 0.1612449
```

```
print(model_tuning$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train_data,
##     ranges = list(0.01, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  345
```

Best Value for optimal cost is 0.1612449

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

Computing with the cost of 0.1612449 and predicting the test and the train error

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'linear', cost = 0.1612449)
```

Getting the summary of the model

```
summary(model_fit)
```

```
## 
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "linear", 
##     cost = 0.1612449)
## 
## 
## Parameters:
##    SVM-Type:  C-classification 
##  SVM-Kernel:  linear 
##        cost:  0.1612449 
## 
## Number of Support Vectors:  357
## 
##  ( 176 181 )
## 
## 
## Number of Classes:  2 
## 
## Levels: 
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"           
##  [5] "degree"          "gamma"           "coef0"           "nu"             
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"        
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"        
## [17] "nSV"             "labels"          "SV"              "index"          
## [21] "rho"             "compprob"        "probA"           "probB"          
## [25] "sigma"           "coefs"           "na.action"       "fitted"         
## [29] "decision.values" "terms"
```

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.5130841
```

Predicting for the new set of values - Test Data

```
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.5831776
```

For Linear Kernel

The training error is 0.5130841 and the test error is 0.5831776

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

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'radial', cost = 0.01)
```

Getting the summary of the model

```
summary(model_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  652
##
##  ( 324 328 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"
##  [5] "degree"          "gamma"           "coef0"           "nu"
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"
## [17] "nSV"             "labels"          "SV"              "index"
## [21] "rho"             "compprob"        "probA"           "probB"
## [25] "sigma"           "coefs"           "na.action"       "fitted"
## [29] "decision.values" "terms"
```

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.6102804
```

Predicting for the new set of values - Test Data

```
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.6102804
```

The training error is 0.6102804 and the test error is 0.6102804

```
model_tuning <- tune(svm, Purchase~., data = train_data, kernel = "radial", ranges = list(0.0
1, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10))
```

Summary of tuned model

```
summary(model_tuning)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.1624624
```

```
names(model_tuning)
```

```
## [1] "best.parameters"  "best.performance" "method"           "nparcomb"
## [5] "train.ind"        "sampling"         "performances"     "best.model"
```

Best value for cost

```
print(model_tuning$best.parameters)
```

```
##   Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
## 1 0.01  0.1  0.5    1  3.5  5.5  7.5   10
```

```
print(model_tuning$best.performance)
```

```
## [1] 0.1624624
```

```
print(model_tuning$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train_data,
##     ranges = list(0.01, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  389
```

Best Value for optimal cost is 0.1624624

Computing with the cost of 0.1624624 and predicting the test and the train error

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'radial', cost = 0.1624624)
```

Getting the summary of the model

```
summary(model_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "radial",
##     cost = 0.1624624)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1624624
##
## Number of Support Vectors:  516
##
##  ( 256 260 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"
##  [5] "degree"          "gamma"           "coef0"           "nu"
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"
## [17] "nSV"             "labels"          "SV"              "index"
## [21] "rho"             "compprob"        "probA"           "probB"
## [25] "sigma"           "coefs"           "na.action"       "fitted"
## [29] "decision.values" "terms"
```

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.5130841
```

Predicting for the new set of values - Test Data

```
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.5841121
```

For Radical Kernel

The training error is 0.5130841 and the test error is 0.5841121

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

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'polynomial', degree = 2, cost = 0.0
1)
```

Getting the summary of the model

```
summary(model_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  653
##
##  ( 324 329 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"
##  [5] "degree"          "gamma"           "coef0"           "nu"
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"
## [17] "nSV"             "labels"          "SV"              "index"
## [21] "rho"             "compprob"        "probA"           "probB"
## [25] "sigma"           "coefs"           "na.action"       "fitted"
## [29] "decision.values" "terms"
```

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.6102804
```

Predicting for the new set of values - Test Data

```
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.611215
```

The training error is 0.6102804 and the test error is 0.611215

```
model_tuning <- tune(svm, Purchase~., data = train_data, kernel = "polynomial",degree = 2, ra
nges = list(0.01, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10))
```

Summary of tuned model

```
summary(model_tuning)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.1928591
```

```
names(model_tuning)
```

```
## [1] "best.parameters"  "best.performance" "method"           "nparcomb"
## [5] "train.ind"        "sampling"         "performances"     "best.model"
```

Best value for cost

```
print(model_tuning$best.parameters)
```

```
##   Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
## 1 0.01  0.1  0.5    1  3.5  5.5  7.5   10
```

```
print(model_tuning$best.performance)
```

```
## [1] 0.1928591
```

```
print(model_tuning$best.model)
```

```
## 
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train_data, 
##     ranges = list(0.01, 0.1, 0.5, 1, 3.5, 5.5, 7.5, 10), kernel = "polynomial", 
##     degree = 2)
## 
## 
## Parameters:
##    SVM-Type:  C-classification 
##  SVM-Kernel:  polynomial 
##        cost:  1 
##      degree:  2 
##      coef.0:  0 
## 
## Number of Support Vectors:  472
```

Best Value for optimal cost is 0.1928591

Computing with the cost of 0.1928591 and predicting the test and the train error

```
set.seed(23)
model_fit <- svm(Purchase~., data = train_data, kernel = 'polynomial', degree = 2, cost = 0.1
928591)
```

Getting the summary of the model

```
summary(model_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, kernel = "polynomial",
##     degree = 2, cost = 0.1928591)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.1928591
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  595
##
##  ( 294 301 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
names(model_fit)
```

```
##  [1] "call"            "type"            "kernel"          "cost"
##  [5] "degree"          "gamma"           "coef0"           "nu"
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"
## [17] "nSV"             "labels"          "SV"              "index"
## [21] "rho"             "compprob"        "probA"           "probB"
## [25] "sigma"           "coefs"           "na.action"       "fitted"
## [29] "decision.values" "terms"
```

Calculating for train data

```
y_predict_train <- predict(model_fit, newdata = train_data, type = "response")
```

Predicting Test Error

```
y_predict_train = as.numeric(y_predict_train)-1
prediction_error_test = length(which(y_predict_train == y_test_data))/length(y_test_data)
```

```
## Warning in y_predict_train == y_test_data: longer object length is not a
## multiple of shorter object length
```

```
print(prediction_error_test)
```

```
## [1] 0.5401869
```

Predicting for the new set of values - Test Data

```r
y_predict_test <- predict(model_fit, newdata = test_data, type = "response")
#print(model_predict)
```

Calculating Prediction Error

```r
y_predict_test = as.numeric(y_predict_test)-1
prediction_error_test = length(which(y_predict_test == y_test_data))/length(y_test_data)
print(prediction_error_test)
```

```
## [1] 0.5915888
```

For Polinomial Kernel

The training error is 0.5401869 and the test error is 0.5915888

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

In the above three models, SVM with Linear Kernel provides better prediction than the other two models. Linear model provides train error of 5130841 and a test error of 0.5831776 which is less than the other models.