DSCI5340_HW4_Group 6

Prasanth Gutha, Manasa Shivkar, Fahimeh Asgari, Vijay Ramaraju Jampana

Loading the Packages

```
pacman::p_load(e1071, ggplot2, caret, rmarkdown, corrplot)
search()
```

```
##  [1] ".GlobalEnv"        "package:corrplot"  "package:rmarkdown"
##  [4] "package:caret"     "package:lattice"   "package:ggplot2"
##  [7] "package:e1071"     "package:stats"     "package:graphics"
## [10] "package:grDevices" "package:utils"     "package:datasets"
## [13] "package:methods"   "Autoloads"         "package:base"
```

```
theme_set(theme_classic())
options(digits = 3)
```

1. Create a training data set containing a random sample of 90% of the observations in the "juice2022.csv" data set using the createDataPartition() function. Create a test data set containing the remaining observations.

```
Juice2022.df <- read.csv("juice2022.csv", stringsAsFactors = TRUE)
str(Juice2022.df)
```

```
## 'data.frame':    1000 obs. of  13 variables:
##  $ X              : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Purchase       : Factor w/ 2 levels "FN","MM": 1 2 1 2 2 1 1 2 2 1 ...
##  $ storeID        : int  3 6 2 9 9 5 8 4 2 5 ...
##  $ ListPriceFN    : num  2.71 2.69 2.69 3.17 3.14 3.3 3.08 2.57 2.51 2.79 ...
##  $ ListPriceMM    : num  2.77 3.73 2.99 3.31 2.74 3.36 3.46 3.14 3.63 3.58 ...
##  $ DiscFN         : num  0.1 0.2 0.1 0.2 0.1 0 0.2 0 0.1 0.2 ...
##  $ DiscMM         : num  0 0.2 0.1 0.1 0 0 0.1 0.1 0.2 0.1 ...
##  $ FinalPriceFN   : num  2.61 2.49 2.59 2.97 3.04 3.3 2.88 2.57 2.41 2.59 ...
##  $ FinalPriceMM   : num  2.77 3.53 2.89 3.21 2.74 3.36 3.36 3.04 3.43 3.48 ...
##  $ ListPriceDiffMM: num  0.06 1.04 0.3 0.14 -0.4 ...
##  $ SalePriceDiffMM: num  0.16 1.04 0.3 0.24 -0.3 ...
##  $ PctDiscFN      : num  3.69 7.43 3.72 6.31 3.18 0 6.49 0 3.98 7.17 ...
##  $ PctDiscMM      : num  0 5.36 3.34 3.02 0 0 2.89 3.18 5.51 2.79 ...
```

#creating the training data set and test data set

```r
set.seed(42)
train.index <- createDataPartition(Juice2022.df$Purchase, p = 0.9, list = FALSE)
train_Juice2022 <- Juice2022.df[train.index, ]
test_Juice2022 <- Juice2022.df[-train.index, ]
```

2. Fit an SVM model with a linear kernel to the training data using cost=0.01. Use Purchase as the response and the other variables as predictors in this model. Use the summary() function to produce summary statistics, and describe the results obtained.

```r
set.seed(42)
svm_linear <- svm(Purchase ~., data = train_Juice2022, cost = 0.01, kernel = "linear")
summary(svm_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_Juice2022, cost = 0.01,
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  823
##
##  ( 424 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

```r
pred_svml <- predict(svm_linear, newdata = test_Juice2022)
accuracyl = mean(pred_svml == test_Juice2022$Purchase)
cat("SVM Linear accuracy: ", accuracyl)
```

```
## SVM Linear accuracy:  0.556
```

- Based on the summary of the SVM using linear kernel, we have 823 Support Vectors. 424 Support Vectors are FN(Florida's Natural) and 399 Support Vectors are MM(Minute Maid).

3. What are the training and test error rates?

```r
## Training error
pred_train_svml <- predict(svm_linear, train_Juice2022)
training_error <- mean(pred_train_svml != train_Juice2022$Purchase)
cat("Training Error rate: ", training_error)
```

```
## Training Error rate:  0.443
```

```r
## Test Error
pred_test_svml <- predict(svm_linear, test_Juice2022)
test_error <- mean(pred_test_svml != test_Juice2022$Purchase)
cat("Test Error rate: ", test_error)
```

```
## Test Error rate:  0.444
```

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

```r
set.seed(42)
svml_tune <- tune(svm, Purchase ~., data = train_Juice2022, Kernel = "linear",
                  ranges = list(cost = c(seq(0.01,0.1,by = 0.01),
                                         seq(0.1,1,by = 0.1),
                                         seq(1,10,by = 1))))
summary(svml_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.443
##
## - Detailed performance results:
##     cost error dispersion
## 1   0.01 0.443     0.0609
## 2   0.02 0.443     0.0609
## 3   0.03 0.443     0.0609
## 4   0.04 0.443     0.0609
## 5   0.05 0.443     0.0609
## 6   0.06 0.443     0.0609
## 7   0.07 0.443     0.0609
## 8   0.08 0.443     0.0609
## 9   0.09 0.443     0.0609
## 10  0.10 0.443     0.0609
## 11  0.10 0.443     0.0609
## 12  0.20 0.444     0.0588
```

```
## 13  0.30 0.448     0.0637
## 14  0.40 0.443     0.0649
## 15  0.50 0.451     0.0602
## 16  0.60 0.460     0.0512
## 17  0.70 0.468     0.0478
## 18  0.80 0.472     0.0519
## 19  0.90 0.482     0.0571
## 20  1.00 0.478     0.0558
## 21  1.00 0.478     0.0558
## 22  2.00 0.488     0.0463
## 23  3.00 0.495     0.0502
## 24  4.00 0.493     0.0465
## 25  5.00 0.498     0.0487
## 26  6.00 0.499     0.0554
## 27  7.00 0.497     0.0581
## 28  8.00 0.497     0.0643
## 29  9.00 0.497     0.0562
## 30 10.00 0.501     0.0498
```

```
Optimal_cost <- svml_tune$best.model$cost
summary(Optimal_cost)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.01    0.01    0.01    0.01    0.01    0.01
```

```
svml_tuned <- svml_tune$best.model
summary(svml_tuned)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = train_Juice2022,
##     ranges = list(cost = c(seq(0.01, 0.1, by = 0.01), seq(0.1, 1,
##         by = 0.1), seq(1, 10, by = 1))), Kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  821
##
##  ( 422 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

- The optimal cost using tune function is 0.01

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

```
pred_svm2 <- predict(svml_tuned, newdata = test_Juice2022)
accuracy = mean(pred_svm2 == test_Juice2022$Purchase)
cat("SVM Linear accuracy: ", accuracy)
```

```
## SVM Linear accuracy:  0.556
```

```
svmlt <- svm(Purchase ~ ., data = train_Juice2022, Kernel = "linear",
             cost = Optimal_cost)

## Training error
pred_train_svm2 <- predict(svmlt, train_Juice2022)
training_error <- mean(pred_train_svm2 != train_Juice2022$Purchase)
cat("New Training Error rate: ", training_error)
```

```
## New Training Error rate:  0.443
```

```
## Test Error
pred_test_svm2 <- predict(svmlt, test_Juice2022)
test_error <- mean(pred_test_svm2 != test_Juice2022$Purchase)
cat("New Test Error rate: ", test_error)
```

```
## New Test Error rate:  0.444
```

```
Linear_model <- data.frame("kernel" = "linear", "Cost" = Optimal_cost,
                           "Training Error" = training_error,
                           "Test Error"= test_error)
Linear_model
```

```
##   kernel Cost Training.Error Test.Error
## 1 linear 0.01          0.443      0.444
```

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

```
## Step 2
set.seed(42)
svm_radial <- svm(Purchase ~., data = train_Juice2022, cost = 0.01,
                           kernel = "radial", gamma = 1)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_Juice2022, cost = 0.01,
##     kernel = "radial", gamma = 1)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  881
##
##  ( 482 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

```
pred_svmr <- predict(svm_radial, newdata = test_Juice2022)
accuracy_radial = mean(pred_svmr == test_Juice2022$Purchase)
cat("SVM radial accuracy: ", accuracy_radial)
```

```
## SVM radial accuracy:  0.556
```

- SVM using radial kernel has 881 support vectors, of which 482 were FN and 399 were MM

```
## Training error
pred_train_svmradial <- predict(svm_radial, train_Juice2022)
training_error <- mean(pred_train_svmradial != train_Juice2022$Purchase)
cat("Training Error rate: ", training_error)
```

```
## Training Error rate:  0.443
```

```
## Test Error
pred_test_svmradial <- predict(svm_radial, test_Juice2022)
test_error <- mean(pred_test_svmradial != test_Juice2022$Purchase)
cat("Test Error rate: ", test_error)
```

```
## Test Error rate:  0.444
```

```
set.seed(42)
svmradial_tune <- tune(svm, Purchase ~., data = train_Juice2022,
                       kernel = "radial", gamma = 1,
                       ranges = list(cost = c(seq(0.01,0.1,by = 0.01),
                                  seq(0.1,1,by = 0.1),
                                  seq(1,10,by = 1))))
summary(svmradial_tune)
```

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

```
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.441
##
## - Detailed performance results:
##     cost error dispersion
## 1   0.01 0.443     0.0609
## 2   0.02 0.443     0.0609
## 3   0.03 0.443     0.0609
## 4   0.04 0.443     0.0609
## 5   0.05 0.443     0.0609
## 6   0.06 0.443     0.0609
## 7   0.07 0.443     0.0609
## 8   0.08 0.443     0.0609
## 9   0.09 0.443     0.0609
## 10  0.10 0.443     0.0609
## 11  0.10 0.443     0.0609
## 12  0.20 0.443     0.0609
## 13  0.30 0.443     0.0609
## 14  0.40 0.445     0.0651
## 15  0.50 0.441     0.0680
## 16  0.60 0.454     0.0648
## 17  0.70 0.470     0.0552
## 18  0.80 0.481     0.0553
## 19  0.90 0.492     0.0467
## 20  1.00 0.507     0.0425
## 21  1.00 0.507     0.0425
## 22  2.00 0.512     0.0409
## 23  3.00 0.507     0.0380
## 24  4.00 0.519     0.0324
## 25  5.00 0.515     0.0341
## 26  6.00 0.515     0.0303
## 27  7.00 0.516     0.0301
## 28  8.00 0.516     0.0328
## 29  9.00 0.516     0.0340
## 30 10.00 0.516     0.0332
```

```
Optimal_costr <- svmradial_tune$best.model$cost
summary(Optimal_cost)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.01    0.01    0.01    0.01    0.01    0.01
```

```
svmradial_tuned <- svmradial_tune$best.model
summary(svmradial_tuned)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = train_Juice2022,
##     ranges = list(cost = c(seq(0.01, 0.1, by = 0.01), seq(0.1, 1,
```

```
##         by = 0.1), seq(1, 10, by = 1))), kernel = "radial", gamma = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  885
##
##  ( 486 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

- The optimal cost using tune function is 0.5

```
pred_svmradial <- predict(svmradial_tuned, newdata = test_Juice2022)
accuracy = mean(pred_svmradial == test_Juice2022$Purchase)
cat("SVM radial accuracy: ", accuracy)
```

```
## SVM radial accuracy:  0.556
```

```
svmrt <- svm(Purchase ~ ., data = train_Juice2022, kernel = "radial",
             cost = Optimal_costr)

## Training error
pred_train_svmradial <- predict(svmrt, train_Juice2022)
training_error <- mean(pred_train_svmradial != train_Juice2022$Purchase)
cat("New Training Error rate: ", training_error)
```

```
## New Training Error rate:  0.401
```

```
## Test Error
pred_test_svmradial <- predict(svmrt, test_Juice2022)
test_error <- mean(pred_test_svmradial != test_Juice2022$Purchase)
cat("New Test Error rate: ", test_error)
```

```
## New Test Error rate:  0.434
```

```
Radial_model <- data.frame("kernel" = "Radial", "Cost" = Optimal_costr,
                           "Training Error" = training_error,
                           "Test Error"= test_error)
Radial_model
```

```
##   kernel Cost Training.Error Test.Error
## 1 Radial  0.5          0.401      0.434
```

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

```
set.seed(42)
svm_polynomial <- svm(Purchase ~., data = train_Juice2022, cost = 0.01,
                          kernel = 'polynomial', degree = 2)
summary(svm_polynomial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_Juice2022, cost = 0.01,
##     kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  824
##
##  ( 425 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

```
pred_svm_poly <- predict(svm_polynomial, newdata = test_Juice2022)
accuracy_poly = mean(pred_svm_poly == test_Juice2022$Purchase)
cat("SVM poly accuracy: ", accuracy_poly)
```

```
## SVM poly accuracy:  0.556
```

- SVM using polynomial has 824 support vectors of which 425 are FN and 399 are MM.

```
## Training error
pred_train_svmpoly <- predict(svm_polynomial, train_Juice2022)
training_error <- mean(pred_train_svmpoly != train_Juice2022$Purchase)
cat("Training Error rate: ", training_error)
```

```
## Training Error rate:  0.443
```

```
## Test Error
pred_test_svmpoly <- predict(svm_polynomial, test_Juice2022)
test_error <- mean(pred_test_svmpoly != test_Juice2022$Purchase)
cat("Test Error rate: ", test_error)
```

```
## Test Error rate:  0.444
```

- The svm using polynomial kernel has training Error rate of 0.443 and test Error rate: 0.444

```
set.seed(42)
svmpoly_tune <- tune(svm, Purchase ~., data = train_Juice2022,
                     degree = 2, kernel = "polynomial",
                     ranges = list(cost = c(seq(0.01, 0.1, by = 0.01),
                                            seq(0.1, 1, by = 0.1),
                                            seq(1, 10, by = 1))))
summary(svmpoly_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.443
##
## - Detailed performance results:
##     cost error dispersion
## 1   0.01 0.443     0.0609
## 2   0.02 0.443     0.0609
## 3   0.03 0.443     0.0609
## 4   0.04 0.443     0.0609
## 5   0.05 0.443     0.0609
## 6   0.06 0.443     0.0609
## 7   0.07 0.443     0.0609
## 8   0.08 0.443     0.0609
## 9   0.09 0.443     0.0609
## 10  0.10 0.443     0.0609
## 11  0.10 0.443     0.0609
## 12  0.20 0.443     0.0609
## 13  0.30 0.444     0.0597
## 14  0.40 0.446     0.0575
## 15  0.50 0.445     0.0587
## 16  0.60 0.447     0.0569
## 17  0.70 0.451     0.0542
## 18  0.80 0.451     0.0534
## 19  0.90 0.454     0.0535
## 20  1.00 0.454     0.0535
## 21  1.00 0.454     0.0535
## 22  2.00 0.466     0.0467
## 23  3.00 0.464     0.0461
## 24  4.00 0.467     0.0517
## 25  5.00 0.460     0.0586
## 26  6.00 0.462     0.0550
## 27  7.00 0.464     0.0519
## 28  8.00 0.467     0.0575
```

```
## 29  9.00 0.465     0.0580
## 30 10.00 0.466     0.0532
```

```
Optimal_costp <- svmpoly_tune$best.model$cost
summary(Optimal_cost)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.01    0.01    0.01    0.01    0.01    0.01
```

```
svmpoly_tuned <- svmpoly_tune$best.model
summary(svmpoly_tuned)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = train_Juice2022,
##     ranges = list(cost = c(seq(0.01, 0.1, by = 0.01), seq(0.1, 1,
##         by = 0.1), seq(1, 10, by = 1))), degree = 2, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  824
##
##  ( 425 399 )
##
##
## Number of Classes:  2
##
## Levels:
##  FN MM
```

- The optimal cost using tune function is 0.5

```
svm_cost <- svm(Purchase ~ ., data = train_Juice2022, Kernel = "polynomial",
                cost = Optimal_cost)

pred_svmpoly <- predict(svmpoly_tuned, newdata = test_Juice2022)
accuracy = mean(pred_svmpoly == test_Juice2022$Purchase)
cat("SVM polynomial accuracy: ", accuracy)
```

```
## SVM polynomial accuracy:  0.556
```

```
svmpt <- svm(Purchase ~ ., data = train_Juice2022, kernel = "polynomial",
             cost = Optimal_costp)

## Training error
```

```
pred_train_svmpoly <- predict(svm_cost, train_Juice2022)
training_error <- mean(pred_train_svmpoly != train_Juice2022$Purchase)
cat("New Training Error rate: ", training_error)
```

```
## New Training Error rate:  0.443
```

```
## Test Error
pred_test_svmpoly <- predict(svm_cost, test_Juice2022)
test_error <- mean(pred_test_svmpoly != test_Juice2022$Purchase)
cat("New Test Error rate: ", test_error)
```

```
## New Test Error rate:  0.444
```

```
Polynomial_model <- data.frame("kernel" = "Polynomial",
                               "Cost" = Optimal_costr,
                               "Training Error" = training_error,
                               "Test Error"= test_error)
Polynomial_model
```

```
##       kernel Cost Training.Error Test.Error
## 1 Polynomial  0.5          0.443      0.444
```

8. Overall, which of the above approaches give the best results using this data? Explain your answer.

```
Best_model <- rbind(Linear_model, Radial_model, Polynomial_model)
Best_model
```

```
##       kernel Cost Training.Error Test.Error
## 1     linear 0.01          0.443      0.444
## 2     Radial 0.50          0.401      0.434
## 3 Polynomial 0.50          0.443      0.444
```

- The svm model with radial kernel gives the best results with the lowest test error rate of 0.434 with an optimal cost of 0.5 as compared to other svm model with linear and polynomial kernels.
- The training error rate and test error rate for svm model with radial kernel are 0.443 and 0.444 before tuning. after tuning, the training error rate and test error rate are 0.401 and 0.434 which shows a bit improvement.