Support Vector Machines

Dmitrii Dunin, ITU ID 94739

International Technological University

Question 1

```
> rm(list = ls())
> setwd("F:/Workspace/R/Homework6")
> par(mar = rep(2, 4))
> wine <- read.csv("winequality-red.csv", header = TRUE, sep = ";")
> library(e1071)
> str(wine$quality)
```

```
 int [1:1599] 5 5 5 6 5 5 5 7 7 5 ...
```

```
> x <- subset(wine, select = -quality)
> y <- as.numeric(wine$quality)
> wine_factor <- cbind(x, quality = as.factor(y))
> wineTrain <- wine_factor[1:1400, ]
> wineTest <- wine_factor[1401:1599, ]
```

1) In a past homework, you performed ridge regression on the wine quality data set. Now use a support vector machine to classify these data.

1a) First classify the data treating the last column as an ordered factor (the wine tasters score). Next treat the last column as a numeric. Which SVM implementation is better? Why do you think it is better?

1b) Using the best version choose two attributes and a slice through the data to plot. Choose a different set of attributes and another set of slices to plot.

1c) Compare and contrast the best version of the SVM with the ridge regression model.

Answer 1

```
> str(wine_factor)
```

```
'data.frame':        1599 obs. of  12 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ..
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
```

```
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : Factor w/ 6 levels "3","4","5","6",..: 3 3 3 4 3 3 3 5 5 3 ...
```

```
> x_factor <- subset(wineTest, select = -quality)
> y_factor <- wineTest$quality
> wine_svm <- svm(quality ~ ., data = wineTrain)
> summary(wine_svm)
```

```
Call:
svm(formula = quality ~ ., data = wineTrain)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  1166

 ( 430 496 172 46 15 7 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8
```

```
> wine_factor_predict <- predict(wine_svm, x_factor)
> 1 - sum(wine_factor_predict == y_factor) / length(y_factor)
```

```
[1] 0.4371859
```

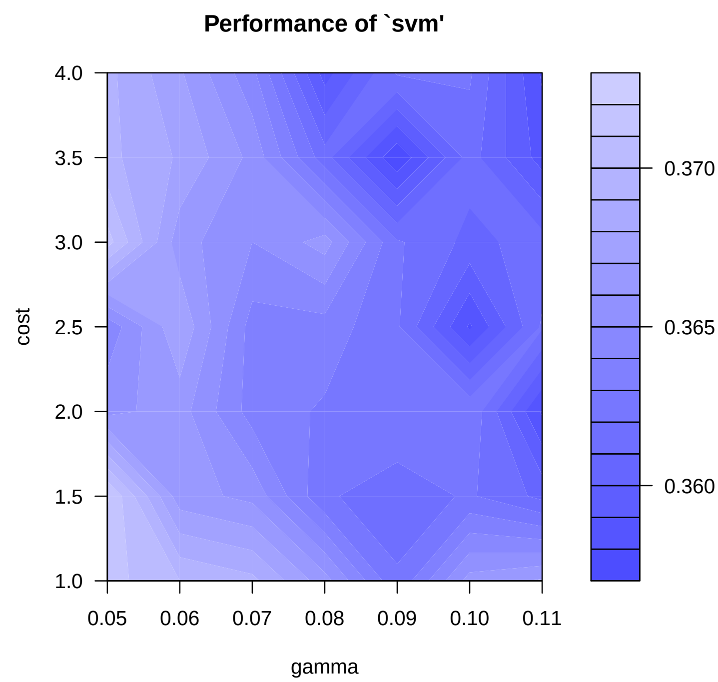
```
> wine_svm_tuned <- tune(
+   svm,
+   quality ~ .,
+   data = wineTrain,
+   ranges = list(gamma = seq(.05, .11, .01), cost = seq(1, 4, 0.5)),
+   tunecontrol = tune.control(sampling = "cross")
+ )
> summary(wine_svm_tuned)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
 gamma cost
  0.09  3.5

- best performance: 0.3571429

- Detailed performance results:
   gamma cost     error dispersion
1   0.05  1.0 0.3714286 0.04504973
2   0.06  1.0 0.3700000 0.04771419
3   0.07  1.0 0.3692857 0.04702607
4   0.08  1.0 0.3664286 0.04960730
5   0.09  1.0 0.3621429 0.04678435
6   0.10  1.0 0.3664286 0.04454990
7   0.11  1.0 0.3671429 0.04405727
8   0.05  1.5 0.3721429 0.04999433
9   0.06  1.5 0.3664286 0.04972145
10  0.07  1.5 0.3657143 0.04806930
11  0.08  1.5 0.3621429 0.04702607
12  0.09  1.5 0.3614286 0.04288359
13  0.10  1.5 0.3621429 0.04110851
14  0.11  1.5 0.3607143 0.04195479
15  0.05  2.0 0.3657143 0.04946425
16  0.06  2.0 0.3664286 0.04926328
17  0.07  2.0 0.3635714 0.04408943
18  0.08  2.0 0.3628571 0.04309458
19  0.09  2.0 0.3628571 0.04162243
20  0.10  2.0 0.3628571 0.03981249
21  0.11  2.0 0.3578571 0.04046919
22  0.05  2.5 0.3642857 0.04773795
23  0.06  2.5 0.3678571 0.04533198
24  0.07  2.5 0.3635714 0.04548180
25  0.08  2.5 0.3635714 0.04408943
26  0.09  2.5 0.3621429 0.04403797
27  0.10  2.5 0.3578571 0.04060903
28  0.11  2.5 0.3621429 0.03810268
29  0.05  3.0 0.3714286 0.04821061
30  0.06  3.0 0.3664286 0.04678435
31  0.07  3.0 0.3650000 0.04370199
32  0.08  3.0 0.3664286 0.04219730
33  0.09  3.0 0.3621429 0.04027260
34  0.10  3.0 0.3607143 0.03842860
35  0.11  3.0 0.3614286 0.03437159
36  0.05  3.5 0.3692857 0.04654138
37  0.06  3.5 0.3678571 0.04545686
38  0.07  3.5 0.3657143 0.04283068
```

```
39  0.08  3.5 0.3614286 0.04085259
40  0.09  3.5 0.3571429 0.03734378
41  0.10  3.5 0.3614286 0.03144300
42  0.11  3.5 0.3585714 0.03190832
43  0.05  4.0 0.3692857 0.04738634
44  0.06  4.0 0.3671429 0.04221745
45  0.07  4.0 0.3642857 0.04219058
46  0.08  4.0 0.3585714 0.03592004
47  0.09  4.0 0.3621429 0.03750283
48  0.10  4.0 0.3621429 0.03086985
49  0.11  4.0 0.3578571 0.02765392
```

```
> plot(wine_svm_tuned)
```



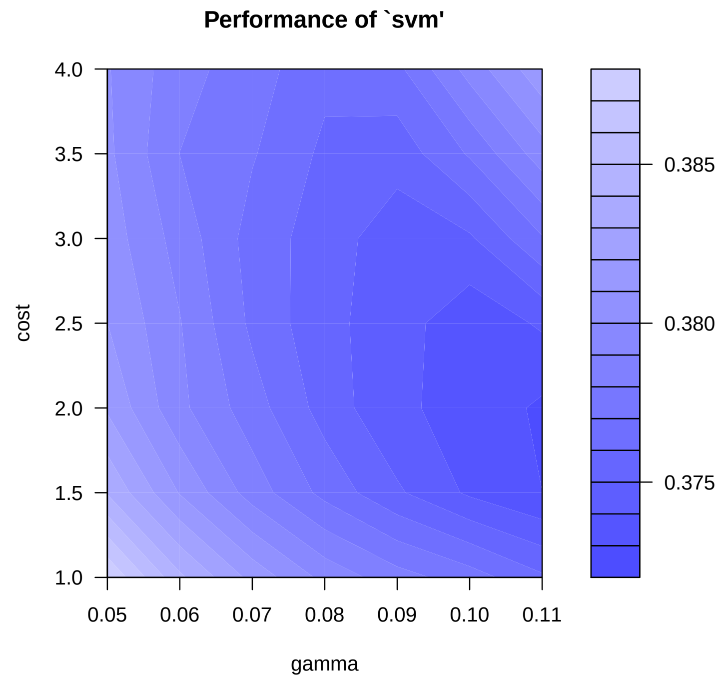

```
> wine_svm_tuned$best.parameters

   gamma cost
40  0.09  3.5

> wine_svm <-
+   svm(quality ~ .,
+       data = wineTrain,
+       gamma = 0.07,
```

```
+       cost = 1.5)
> wine_factor_predict <- predict(wine_svm, x_factor)
> 1 - sum(wine_factor_predict == y_factor) / length(y_factor)

[1] 0.4371859

> wine_numeric <- cbind(x, quality = y)
> str(wine_numeric)

'data.frame':        1599 obs. of  12 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ..
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : num  5 5 5 6 5 5 5 7 7 5 ...

> wineTrain <- wine_numeric[1:1400,]
> wineTest <- wine_numeric[1401:1599,]
> x_factor <- subset(wineTest, select = -quality)
> y_factor <- wineTest$quality
> wine_svm <- svm(quality ~ ., data = wineTrain)
> summary(wine_svm)

Call:
svm(formula = quality ~ ., data = wineTrain)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.09090909
    epsilon:  0.1


Number of Support Vectors:  1162

> wine_factor_predict <- predict(wine_svm, x_factor)
> sqrt(sum((wineTest$quality - wine_factor_predict) ^ 2)) / length(wine_factor_predict)

[1] 0.04847373
```

```
> wine_svm_tuned <- tune(
+   svm,
+   quality ~ .,
+   data = wineTrain,
+   ranges = list(gamma = seq(.05, .11, .01), cost = seq(1, 4, 0.5)),
+   tunecontrol = tune.control(sampling = "cross")
+ )
> summary(wine_svm_tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.11    2

- best performance: 0.3727997

- Detailed performance results:
   gamma cost     error dispersion
1   0.05  1.0 0.3879719 0.06354646
2   0.06  1.0 0.3842144 0.06359855
3   0.07  1.0 0.3816850 0.06405247
4   0.08  1.0 0.3797012 0.06406514
5   0.09  1.0 0.3784169 0.06481624
6   0.10  1.0 0.3774793 0.06552371
7   0.11  1.0 0.3761696 0.06612698
8   0.05  1.5 0.3839198 0.06380552
9   0.06  1.5 0.3809303 0.06500460
10  0.07  1.5 0.3785399 0.06571189
11  0.08  1.5 0.3766935 0.06588932
12  0.09  1.5 0.3751538 0.06661224
13  0.10  1.5 0.3738265 0.06698094
14  0.11  1.5 0.3730044 0.06740230
15  0.05  2.0 0.3818595 0.06480226
16  0.06  2.0 0.3792459 0.06662128
17  0.07  2.0 0.3774536 0.06690832
18  0.08  2.0 0.3755914 0.06752239
19  0.09  2.0 0.3741436 0.06765951
20  0.10  2.0 0.3737118 0.06785875
21  0.11  2.0 0.3727997 0.06782081
22  0.05  2.5 0.3810074 0.06648126
23  0.06  2.5 0.3790542 0.06773509
24  0.07  2.5 0.3767841 0.06798148
25  0.08  2.5 0.3752818 0.06843633
```

```
26  0.09  2.5 0.3744584 0.06811037
27  0.10  2.5 0.3732920 0.06760679
28  0.11  2.5 0.3741486 0.06769020
29  0.05  3.0 0.3805366 0.06759578
30  0.06  3.0 0.3785949 0.06847822
31  0.07  3.0 0.3765961 0.06880767
32  0.08  3.0 0.3754720 0.06839257
33  0.09  3.0 0.3744464 0.06769315
34  0.10  3.0 0.3748511 0.06765467
35  0.11  3.0 0.3769096 0.06798540
36  0.05  3.5 0.3802149 0.06858679
37  0.06  3.5 0.3779904 0.06892722
38  0.07  3.5 0.3770890 0.06884818
39  0.08  3.5 0.3757915 0.06817925
40  0.09  3.5 0.3753924 0.06718889
41  0.10  3.5 0.3771181 0.06779800
42  0.11  3.5 0.3795433 0.06790238
43  0.05  4.0 0.3800792 0.06889037
44  0.06  4.0 0.3783843 0.06973944
45  0.07  4.0 0.3774527 0.06873261
46  0.08  4.0 0.3762711 0.06758190
47  0.09  4.0 0.3767484 0.06748039
48  0.10  4.0 0.3794037 0.06746217
49  0.11  4.0 0.3817148 0.06810126
```

```
> plot(wine_svm_tuned)
```




```
> wine_svm_tuned$best.parameters

   gamma cost
21  0.11    2

> wine_svm <-
+   svm(quality ~ .,
+       data = wineTrain,
+       gamma = 0.1,
+       cost = 2)
> wine_factor_predict <- predict(wine_svm, x_factor)
> sqrt(sum((wineTest$quality - wine_factor_predict) ^ 2)) / length(wine_factor_predict)

[1] 0.04952251

> print("quality as factor had error = 0.437 but numeric quality had small error = 0.0495")

[1] "quality as factor had error = 0.437 but numeric quality had small error = 0.0495"

> print("Regression is better than classification")

[1] "Regression is better than classification"
```

Question 2

```
> rm(list = ls())
> sonarTest <- read.csv("sonar_test.csv", header = FALSE)
> sonarTest$V61[sonarTest$V61 == -1] <- 0
> sonarTrain <- read.csv("sonar_train.csv", header = FALSE)
> sonarTrain$V61[sonarTrain$V61 == -1] <- 0
> x <- subset(sonarTest, select = -V61)
> y <- sonarTest$V61
> sonar_svm <- svm(V61 ~ ., data = sonarTrain)
> summary(sonar_svm)

Call:
svm(formula = V61 ~ ., data = sonarTrain)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.01666667
    epsilon:  0.1


Number of Support Vectors:  111
```

2) Classify the sonar data set.

2a) Use a support vector machine to classify the sonar data set. First tune an SVM employing radial basis function (default). Next tune an SVM employing a linear kernel. Compare the results.

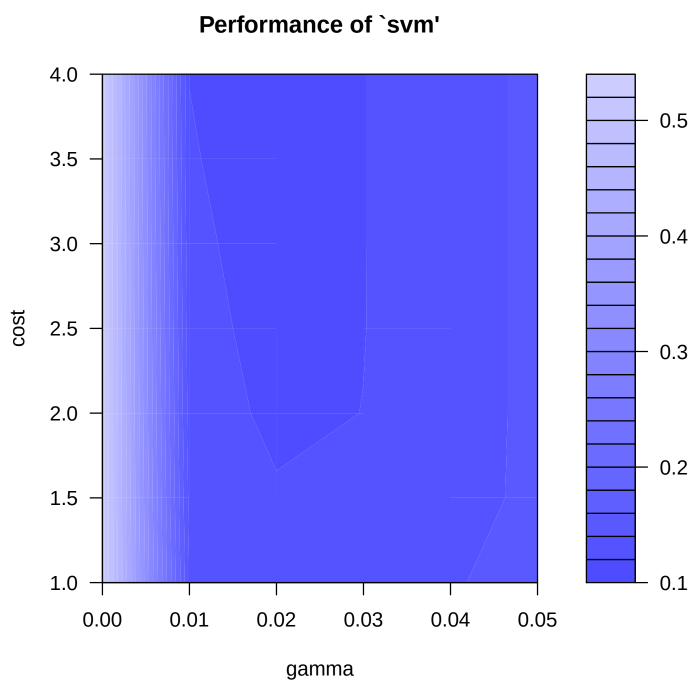
Answer 2

```
> sonar_predict <- predict(sonar_svm, x)
> sqrt(sum((y - sonar_predict) ^ 2)) / length(sonarTest)

[1] 0.05062342

> sonar_svm_tuned <- tune(
+   svm,
+   V61 ~ .,
+   data = sonarTrain,
+   ranges = list(gamma = seq(0, .05, .01), cost = seq(1, 4, 0.5)),
+   tunecontrol = tune.control(sampling = "cross")
+ )
> summary(sonar_svm_tuned)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.02    4

- best performance: 0.1096252

- Detailed performance results:
   gamma cost     error dispersion
1   0.00  1.0 0.5284534 0.08754793
2   0.01  1.0 0.1394513 0.04015638
3   0.02  1.0 0.1293766 0.03578565
4   0.03  1.0 0.1312257 0.02904817
5   0.04  1.0 0.1383221 0.02571119
6   0.05  1.0 0.1473151 0.02427383
7   0.00  1.5 0.5284534 0.08754793
8   0.01  1.5 0.1339546 0.04181752
9   0.02  1.5 0.1219896 0.03235760
10  0.03  1.5 0.1233045 0.02610493
11  0.04  1.5 0.1329505 0.02432079
12  0.05  1.5 0.1441553 0.02370509
13  0.00  2.0 0.5284534 0.08754793
14  0.01  2.0 0.1300873 0.04241755
15  0.02  2.0 0.1157550 0.02976136
16  0.03  2.0 0.1202153 0.02555364
17  0.04  2.0 0.1321375 0.02414856
18  0.05  2.0 0.1441243 0.02369140
19  0.00  2.5 0.5284534 0.08754793
20  0.01  2.5 0.1271870 0.04103829
21  0.02  2.5 0.1128271 0.02919698
22  0.03  2.5 0.1195963 0.02559356
23  0.04  2.5 0.1321384 0.02414786
24  0.05  2.5 0.1441243 0.02369140
25  0.00  3.0 0.5284534 0.08754793
26  0.01  3.0 0.1244914 0.03981698
27  0.02  3.0 0.1107846 0.02918212
28  0.03  3.0 0.1195709 0.02558913
29  0.04  3.0 0.1321384 0.02414786
30  0.05  3.0 0.1441243 0.02369140
31  0.00  3.5 0.5284534 0.08754793
32  0.01  3.5 0.1215761 0.03818792
33  0.02  3.5 0.1098412 0.02940342
34  0.03  3.5 0.1195709 0.02558913
```

```
35  0.04  3.5 0.1321384 0.02414786
36  0.05  3.5 0.1441243 0.02369140
37  0.00  4.0 0.5284534 0.08754793
38  0.01  4.0 0.1196551 0.03706840
39  0.02  4.0 0.1096252 0.02942484
40  0.03  4.0 0.1195709 0.02558913
41  0.04  4.0 0.1321384 0.02414786
42  0.05  4.0 0.1441243 0.02369140
```

```
> plot(sonar_svm_tuned)
```




```
> sonar_svm_tuned$best.parameters
```

```
   gamma cost
39  0.02    4
```

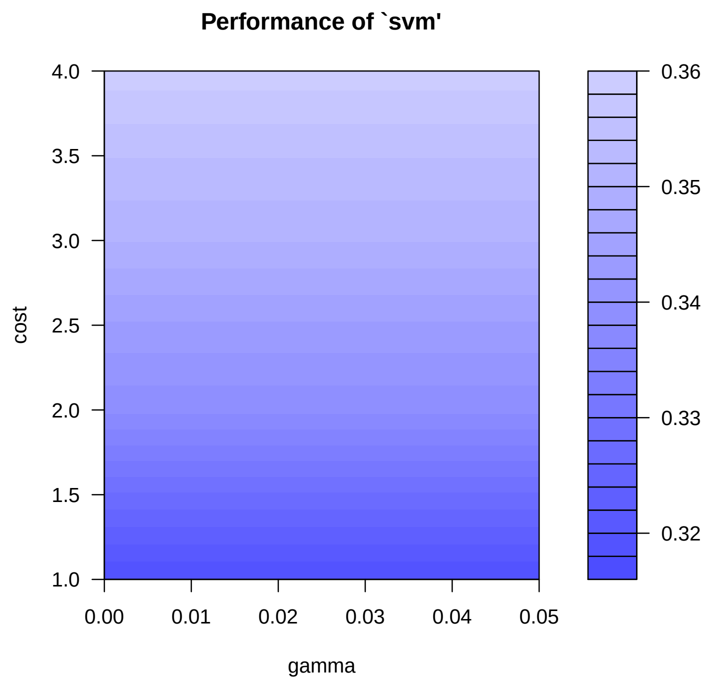
```
> sonar_svm <- svm(V61 ~ .,
+                  data = sonarTrain,
+                  gamma = 0.02,
+                  cost = 4)
> sonar_predict <- predict(sonar_svm, x)
> sqrt(sum((y - sonar_predict) ^ 2)) / length(y)
```

```
[1] 0.03728963
```

```
> sonar_svm <- svm(V61 ~ ., data = sonarTrain, kernel = "linear")
> summary(sonar_svm)

Call:
svm(formula = V61 ~ ., data = sonarTrain, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.01666667
    epsilon:  0.1


Number of Support Vectors:  120

> sonar_svm_tuned <-
+   tune(
+     svm,
+     V61 ~ .,
+     data = sonarTrain,
+     kernel = "linear",
+     ranges = list(gamma = seq(0, .05, .01), cost = seq(1, 4, 0.5)),
+     tunecontrol = tune.control(sampling = "cross")
+   )
> summary(sonar_svm_tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
     0    1

- best performance: 0.3179544

- Detailed performance results:
   gamma cost     error dispersion
1   0.00  1.0 0.3179544  0.1749341
2   0.01  1.0 0.3179544  0.1749341
3   0.02  1.0 0.3179544  0.1749341
4   0.03  1.0 0.3179544  0.1749341
5   0.04  1.0 0.3179544  0.1749341
6   0.05  1.0 0.3179544  0.1749341
7   0.00  1.5 0.3277017  0.1840205
```

```
8   0.01  1.5 0.3277017  0.1840205
9   0.02  1.5 0.3277017  0.1840205
10  0.03  1.5 0.3277017  0.1840205
11  0.04  1.5 0.3277017  0.1840205
12  0.05  1.5 0.3277017  0.1840205
13  0.00  2.0 0.3384717  0.1913417
14  0.01  2.0 0.3384717  0.1913417
15  0.02  2.0 0.3384717  0.1913417
16  0.03  2.0 0.3384717  0.1913417
17  0.04  2.0 0.3384717  0.1913417
18  0.05  2.0 0.3384717  0.1913417
19  0.00  2.5 0.3437060  0.1917421
20  0.01  2.5 0.3437060  0.1917421
21  0.02  2.5 0.3437060  0.1917421
22  0.03  2.5 0.3437060  0.1917421
23  0.04  2.5 0.3437060  0.1917421
24  0.05  2.5 0.3437060  0.1917421
25  0.00  3.0 0.3501144  0.1916731
26  0.01  3.0 0.3501144  0.1916731
27  0.02  3.0 0.3501144  0.1916731
28  0.03  3.0 0.3501144  0.1916731
29  0.04  3.0 0.3501144  0.1916731
30  0.05  3.0 0.3501144  0.1916731
31  0.00  3.5 0.3540963  0.1918345
32  0.01  3.5 0.3540963  0.1918345
33  0.02  3.5 0.3540963  0.1918345
34  0.03  3.5 0.3540963  0.1918345
35  0.04  3.5 0.3540963  0.1918345
36  0.05  3.5 0.3540963  0.1918345
37  0.00  4.0 0.3591636  0.1975591
38  0.01  4.0 0.3591636  0.1975591
39  0.02  4.0 0.3591636  0.1975591
40  0.03  4.0 0.3591636  0.1975591
41  0.04  4.0 0.3591636  0.1975591
42  0.05  4.0 0.3591636  0.1975591
```

```
> plot(sonar_svm_tuned)
```




```
> sonar_svm_tuned$best.parameters

  gamma cost
1     0    1

> sonar_svm <-
+   svm(
+     V61 ~ .,
+     data = sonarTrain,
+     gamma = 0,
+     cost = 1,
+     kernel = "linear"
+   )
> sonar_predict <- predict(sonar_svm, x)
> error <- sqrt(sum((y - sonar_predict) ^ 2)) / length(y)
> print("In Homework 2 Problem 4 Sonar Test Error using trees was 0.2564103")

[1] "In Homework 2 Problem 4 Sonar Test Error using trees was 0.2564103"

> paste("Smaller Sonar Test Error using SVM was", error)

[1] "Smaller Sonar Test Error using SVM was 0.0581468133184799"
```

Question 3

3) The in class example (svm1.r) used the glass data set. Use the Random Forest technique on the glass data. Compare the Random Forest results with the results obtained in class with SVM

```
> library(randomForest)
> library(mlbench)
> data(Glass, package = "mlbench")
> str(Glass)
```

```
'data.frame':        214 obs. of  10 variables:
 $ RI  : num  1.52 1.52 1.52 1.52 1.52 ...
 $ Na  : num  13.6 13.9 13.5 13.2 13.3 ...
 $ Mg  : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
 $ Al  : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
 $ Si  : num  71.8 72.7 73 72.6 73.1 ...
 $ K   : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
 $ Ca  : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
 $ Ba  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Fe  : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
 $ Type: Factor w/ 6 levels "1","2","3","5",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Answer 3

```
> Glass$Type
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3
[149] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6
[186] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
Levels: 1 2 3 5 6 7
```

```
> index <- 1:nrow(Glass)
> set.seed(pi)
> testindex <- sample(index, trunc(length(index) / 3))
> testset <- Glass[testindex,]
> trainset <- Glass[-testindex,]
> x <- subset(trainset, select = -Type)
> y <- trainset$Type
> rf_Glass_Model <- randomForest(x, y)
> xTest <- subset(testset, select = -Type)
> yTest <- testset$Type
> predictGlass <- predict(rf_Glass_Model, xTest)
```

```
> error <- 1 - sum(predictGlass == yTest) / length(yTest)
> paste("Random Forest, with seed = pi, error =", error)
```

```
[1] "Random Forest, with seed = pi, error = 0.253521126760563"
```

Question 4

4) Choose a new data set which we haven't used in class yet (suggestion: choose one from http://archive.ics.uci.edu/ml/.) Use SVM to classify the data set. Try different kernels. Does changing the kernel make a difference? Which kernel resulted in the smallest error? Use another technique to classify the data set. Which resulted in the better model? (Make sure you describe the data set)

```
> rm(list = ls())
> abalone <- read.csv(file = "abalone.data", header = FALSE)
> head(abalone)
```

```
  V1    V2    V3    V4     V5     V6     V7    V8 V9
1  M 0.455 0.365 0.095 0.5140 0.2245 0.1010 0.150 15
2  M 0.350 0.265 0.090 0.2255 0.0995 0.0485 0.070  7
3  F 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.210  9
4  M 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.155 10
5  I 0.330 0.255 0.080 0.2050 0.0895 0.0395 0.055  7
6  I 0.425 0.300 0.095 0.3515 0.1410 0.0775 0.120  8
```

```
> colnames(abalone) <-
+   c(
+     "Sex",
+     "Length",
+     "Diameter",
+     "Height",
+     "Whole weight",
+     "Shucked weight",
+     "Viscera weight",
+     "Shell weight",
+     "Rings"
+   )
> str(abalone)
```

```
'data.frame':        4177 obs. of  9 variables:
 $ Sex           : Factor w/ 3 levels "F","I","M": 3 3 1 3 2 2 1 1 3 1 ...
 $ Length        : num  0.455 0.35 0.53 0.44 0.33 0.425 0.53 0.545 0.475 0.55 ...
 $ Diameter      : num  0.365 0.265 0.42 0.365 0.255 0.3 0.415 0.425 0.37 0.44 ...
 $ Height        : num  0.095 0.09 0.135 0.125 0.08 0.095 0.15 0.125 0.125 0.15 ...
 $ Whole weight  : num  0.514 0.226 0.677 0.516 0.205 ...
 $ Shucked weight: num  0.2245 0.0995 0.2565 0.2155 0.0895 ...
```

```
 $ Viscera weight: num  0.101 0.0485 0.1415 0.114 0.0395 ...
 $ Shell weight  : num  0.15 0.07 0.21 0.155 0.055 0.12 0.33 0.26 0.165 0.32 ...
 $ Rings         : int  15 7 9 10 7 8 20 16 9 19 ...
```

Answer 4

```
> abaloneTrain <- abalone[1:2500, ]
> abaloneTest <- abalone[2501:4177, ]
> x <- subset(abaloneTest, select = -Rings)
> y <- abaloneTest$Rings
> abalone_svm <- svm(Rings ~ ., data = abaloneTrain)
> summary(abalone_svm)

Call:
svm(formula = Rings ~ ., data = abaloneTrain)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1
    epsilon:  0.1


Number of Support Vectors:  2045

> abalone_predict <- predict(abalone_svm, x)
> sqrt(sum((y - abalone_predict) ^ 2)) / length(y)

[1] 0.04347119

> abalone_svm_tuned <- tune(
+   svm,
+   Rings ~ .,
+   data = abaloneTrain,
+   ranges = list(gamma = seq(0, .4, .1), cost = seq(1, 3, 0.5)),
+   tunecontrol = tune.control(sampling = "cross")
+ )
> summary(abalone_svm_tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
```

```
 gamma cost
   0.1    3

- best performance: 5.308015

- Detailed performance results:
   gamma cost     error dispersion
1    0.0  1.0 12.413664  1.1924080
2    0.1  1.0  5.345207  0.5898319
3    0.2  1.0  5.378102  0.6377388
4    0.3  1.0  5.438773  0.6772802
5    0.4  1.0  5.512487  0.7173763
6    0.0  1.5 12.413664  1.1924080
7    0.1  1.5  5.318594  0.5908056
8    0.2  1.5  5.371063  0.6356439
9    0.3  1.5  5.452387  0.6719794
10   0.4  1.5  5.549601  0.7144408
11   0.0  2.0 12.413664  1.1924080
12   0.1  2.0  5.316211  0.5980449
13   0.2  2.0  5.384165  0.6386219
14   0.3  2.0  5.484174  0.6748355
15   0.4  2.0  5.584636  0.7020716
16   0.0  2.5 12.413664  1.1924080
17   0.1  2.5  5.312537  0.5897154
18   0.2  2.5  5.403690  0.6413851
19   0.3  2.5  5.519046  0.6742353
20   0.4  2.5  5.615734  0.6884678
21   0.0  3.0 12.413664  1.1924080
22   0.1  3.0  5.308015  0.5877680
23   0.2  3.0  5.424777  0.6406752
24   0.3  3.0  5.552888  0.6744293
25   0.4  3.0  5.644935  0.6695168

> abalone_svm_tuned$best.parameters

   gamma cost
22   0.1    3

> abalone_svm <-
+   svm(Rings ~ .,
+       data = abaloneTrain,
+       gamma = 0.1,
+       cost = 2)
> abalone_predict <- predict(abalone_svm, x)
> rb_error <- sqrt(sum((y - abalone_predict) ^ 2)) / length(y)
> rb_error
```

```
[1] 0.04341996

> abalone_svm <- svm(Rings ~ ., data = abaloneTrain, kernel = "linear")
> summary(abalone_svm)

Call:
svm(formula = Rings ~ ., data = abaloneTrain, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.1
    epsilon:  0.1


Number of Support Vectors:  2082

> abalone_predict <- predict(abalone_svm, x)
> sqrt(sum((y - abalone_predict) ^ 2)) / length(y)

[1] 0.0471688

> abalone_svm <-
+   svm(Rings ~ ., data = abaloneTrain, kernel = "polynomial")
> summary(abalone_svm)

Call:
svm(formula = Rings ~ ., data = abaloneTrain, kernel = "polynomial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  0.1
     coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  2083

> abalone_predict <- predict(abalone_svm, x)
> sqrt(sum((y - abalone_predict) ^ 2)) / length(y)

[1] 0.04901388
```

```
> library(randomForest)
> xTrain <- subset(abaloneTrain, select = -Rings)
> yTrain <- abaloneTrain$Rings
> rf_Abalone_Model <- randomForest(xTrain, yTrain)
> predictAbalone <- predict(rf_Abalone_Model, x)
> sqrt(sum((y - predictAbalone) ^ 2)) / length(y)

[1] 0.04636412

> print("SVM with Radial basis kernel is better model")

[1] "SVM with Radial basis kernel is better model"
```

Question 5

5) Use SVM with kernel = "linear" to create regression predictions on the data set created using these lines of code: x <- seq(0.1, 5, by = 0.05) # the observed feature y <- log(x) + rnorm(x, sd = 0.2) # the target for the observed feature Next try various kernels and added features with SVM. Can you improve the model by adding an extra feature which might be a function of the first feature? Compare both lm.ridge and svm. Which method produced a better model? (don't forget to tune your models)

```
> rm(list = ls())
> x <- seq(0.1, 5, by = 0.05)
> y <- log(x) + rnorm(x, sd = 0.2)
> dataset <- as.data.frame(cbind(x, y))
> str(dataset)

'data.frame':	99 obs. of  2 variables:
 $ x: num  0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 ...
 $ y: num  -2.56 -1.72 -1.92 -1.25 -1.23 ...
```

Answer 5

```
> dataset_svm <- svm(y ~ ., data = dataset, kernel = "linear")
> summary(dataset_svm)

Call:
svm(formula = y ~ ., data = dataset, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
```

```
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  77
```

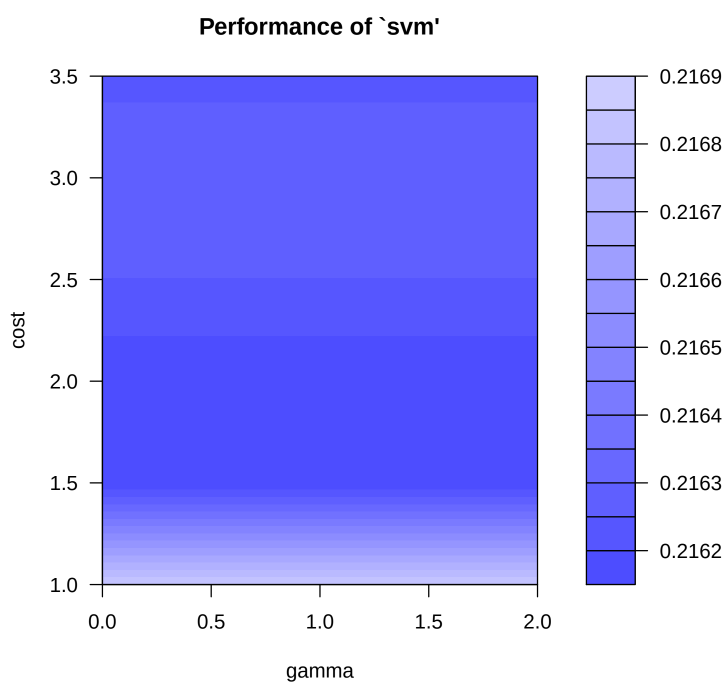
```
> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)
```

```
[1] 0.04587578
```

```
> dataset_svm_tuned <-
+   tune(
+     svm,
+     y ~ .,
+     data = dataset,
+     kernel = "linear",
+     ranges = list(gamma = seq(0, 2, .5), cost = seq(1, 3.5, 0.5)),
+     tunecontrol = tune.control(sampling = "cross")
+   )
> summary(dataset_svm_tuned)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
     0  1.5

- best performance: 0.216152

- Detailed performance results:
   gamma cost     error dispersion
1    0.0  1.0 0.2168505  0.2534500
2    0.5  1.0 0.2168505  0.2534500
3    1.0  1.0 0.2168505  0.2534500
4    1.5  1.0 0.2168505  0.2534500
5    2.0  1.0 0.2168505  0.2534500
6    0.0  1.5 0.2161520  0.2518598
7    0.5  1.5 0.2161520  0.2518598
8    1.0  1.5 0.2161520  0.2518598
9    1.5  1.5 0.2161520  0.2518598
10   2.0  1.5 0.2161520  0.2518598
11   0.0  2.0 0.2161603  0.2518527
12   0.5  2.0 0.2161603  0.2518527
13   1.0  2.0 0.2161603  0.2518527
```

```
14   1.5  2.0 0.2161603  0.2518527
15   2.0  2.0 0.2161603  0.2518527
16   0.0  2.5 0.2162496  0.2517915
17   0.5  2.5 0.2162496  0.2517915
18   1.0  2.5 0.2162496  0.2517915
19   1.5  2.5 0.2162496  0.2517915
20   2.0  2.5 0.2162496  0.2517915
21   0.0  3.0 0.2162735  0.2518607
22   0.5  3.0 0.2162735  0.2518607
23   1.0  3.0 0.2162735  0.2518607
24   1.5  3.0 0.2162735  0.2518607
25   2.0  3.0 0.2162735  0.2518607
26   0.0  3.5 0.2162417  0.2517963
27   0.5  3.5 0.2162417  0.2517963
28   1.0  3.5 0.2162417  0.2517963
29   1.5  3.5 0.2162417  0.2517963
30   2.0  3.5 0.2162417  0.2517963
> plot(dataset_svm_tuned)
```




```
> dataset_svm_tuned$best.parameters
  gamma cost
6     0  1.5
```

```
> dataset_svm <-
+   svm(
+     y ~ .,
+     data = dataset,
+     kernel = "linear",
+     gamma = 0,
+     cost = 2.5
+   )
> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.04587578

> dataset_svm <- svm(y ~ ., data = dataset)
> summary(dataset_svm)

Call:
svm(formula = y ~ ., data = dataset)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  70

> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.02537258

> dataset_svm_tuned <- tune(
+   svm,
+   y ~ .,
+   data = dataset,
+   ranges = list(gamma = seq(0, 2, .5), cost = seq(1, 3.5, 0.5)),
+   tunecontrol = tune.control(sampling = "cross")
+ )
> dataset_svm_tuned$best.parameters

   gamma cost
29   1.5  3.5
```

```
> dataset_svm <- svm(y ~ .,
+                    data = dataset,
+                    gamma = 1.5,
+                    cost = 3.5)
> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.02231531

> dataset_svm <- svm(y ~ ., data = dataset, kernel = "polynomial")
> summary(dataset_svm)

Call:
svm(formula = y ~ ., data = dataset, kernel = "polynomial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  1
     coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  76

> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.04539632

> dataset_svm_tuned <-
+   tune(
+     svm,
+     y ~ .,
+     data = dataset,
+     kernel = "polynomial",
+     ranges = list(gamma = seq(0, 2, .5), cost = seq(1, 3.5, 0.5)),
+     tunecontrol = tune.control(sampling = "cross")
+   )
> dataset_svm_tuned$best.parameters

  gamma cost
2   0.5    1
```

```
> dataset_svm <-
+   svm(
+     y ~ .,
+     data = dataset,
+     kernel = "polynomial",
+     gamma = 0.5,
+     cost = 1.5
+   )
> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.04529434

> dataset_svm <- svm(y ~ ., data = dataset, kernel = "sigmoid")
> summary(dataset_svm)

Call:
svm(formula = y ~ ., data = dataset, kernel = "sigmoid")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  sigmoid
       cost:  1
      gamma:  1
     coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  99

> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.3485689

> dataset_svm_tuned <-
+   tune(
+     svm,
+     y ~ .,
+     data = dataset,
+     kernel = "sigmoid",
+     ranges = list(gamma = seq(0, 2, .5), cost = seq(1, 3.5, 0.5)),
+     tunecontrol = tune.control(sampling = "cross")
+   )
> dataset_svm_tuned$best.parameters

  gamma cost
1     0    1
```

```
> dataset_svm <-
+   svm(
+     y ~ .,
+     data = dataset,
+     kernel = "sigmoid",
+     gamma = 0,
+     cost = 1
+   )
> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.09295157

> xlog <- log(x)
> dataset <- as.data.frame(cbind(x, xlog, y))
> str(dataset)

'data.frame':        99 obs. of  3 variables:
 $ x   : num  0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 ...
 $ xlog: num  -2.3 -1.9 -1.61 -1.39 -1.2 ...
 $ y   : num  -2.56 -1.72 -1.92 -1.25 -1.23 ...

> x <- dataset[, 1:2]
> dataset_svm <- svm(y ~ ., data = dataset, kernel = "linear")
> summary(dataset_svm)

Call:
svm(formula = y ~ ., data = dataset, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  71

> dataset_predict <- predict(dataset_svm, x)
> sqrt(sum((y - dataset_predict) ^ 2)) / length(y)

[1] 0.02076277

> dataset_svm_tuned <-
+   tune(
```

```
+     svm,
+     y ~ .,
+     data = dataset,
+     kernel = "linear",
+     ranges = list(gamma = seq(0, 2, .5), cost = seq(1, 3.5, 0.5)),
+     tunecontrol = tune.control(sampling = "cross")
+   )
> summary(dataset_svm_tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
     0  2.5

- best performance: 0.04642524

- Detailed performance results:
   gamma cost      error dispersion
1    0.0  1.0 0.04712662 0.02422669
2    0.5  1.0 0.04712662 0.02422669
3    1.0  1.0 0.04712662 0.02422669
4    1.5  1.0 0.04712662 0.02422669
5    2.0  1.0 0.04712662 0.02422669
6    0.0  1.5 0.04720073 0.02420670
7    0.5  1.5 0.04720073 0.02420670
8    1.0  1.5 0.04720073 0.02420670
9    1.5  1.5 0.04720073 0.02420670
10   2.0  1.5 0.04720073 0.02420670
11   0.0  2.0 0.04659926 0.02416516
12   0.5  2.0 0.04659926 0.02416516
13   1.0  2.0 0.04659926 0.02416516
14   1.5  2.0 0.04659926 0.02416516
15   2.0  2.0 0.04659926 0.02416516
16   0.0  2.5 0.04642524 0.02416834
17   0.5  2.5 0.04642524 0.02416834
18   1.0  2.5 0.04642524 0.02416834
19   1.5  2.5 0.04642524 0.02416834
20   2.0  2.5 0.04642524 0.02416834
21   0.0  3.0 0.04645327 0.02418050
22   0.5  3.0 0.04645327 0.02418050
23   1.0  3.0 0.04645327 0.02418050
24   1.5  3.0 0.04645327 0.02418050
25   2.0  3.0 0.04645327 0.02418050
```

```
26   0.0  3.5 0.04644910 0.02417828
27   0.5  3.5 0.04644910 0.02417828
28   1.0  3.5 0.04644910 0.02417828
29   1.5  3.5 0.04644910 0.02417828
30   2.0  3.5 0.04644910 0.02417828
```

```
> dataset_svm_tuned$best.parameters
```

```
   gamma cost
16     0  2.5
```

```
> dataset_svm <-
+   svm(
+     y ~ .,
+     data = dataset,
+     kernel = "linear",
+     gamma = 0,
+     cost = 2.5
+   )
> dataset_predict <- predict(dataset_svm, x)
> svm_error <- sqrt(sum((y - dataset_predict) ^ 2)) / length(y)
> library(glmnet)
> dataset <- cbind(x, xlog, y)
> grid = 10 ^ seq(10, -2, length = 100)
> cv.out = cv.glmnet(as.matrix(dataset), y, alpha = 0, lambda = grid)
> cv.out$lambda.min
```

```
[1] 0.01
```

```
> ridgeMod = glmnet(as.matrix(dataset), y, alpha = 0, lambda = 0.01)
> ridgePredict <- predict(ridgeMod, newx = as.matrix(dataset))
> error <- sqrt(sum((y - ridgePredict) ^ 2)) / length(y)
> paste("Ridge Regression error", error)
```

```
[1] "Ridge Regression error 0.00327076627628471"
```

```
> paste("SVM error", svm_error)
```

```
[1] "SVM error 0.0207696882189039"
```

```
> print("lm.ridge produced a better model than SVM")
```

```
[1] "lm.ridge produced a better model than SVM"
```