SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 8: Support vector machine

DR. Rui Zhu, DR. Feng Zhou

2020-03-10

In R exercise 7, you will know

- How to use SVM in R
- How to tune parameters
- How to SVM in caret
- How to visualise SVM with different combinations of parameters

Don't forget to change your working directory!

1 SVM

The `svm()` function in R is usually used to perform SVM.

```
#install.packages("e1071")
library(e1071)
?svm
```

The following example is from the document of the `e1071` package.

```
data(iris)
attach(iris)
## classification mode
model = svm(iris[,-5], iris[,5])
```

Note that, to use `svm()` for classification, we need to make sure that the response vector is a factor vector.

```
summary(model)
```

```
##
## Call:
## svm.default(x = iris[, -5], y = iris[, 5])
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  51
##
##  ( 8 22 21 )
##
##
## Number of Classes:  3
##
```

```
## Levels:
##  setosa versicolor virginica
```

```
model$gamma
```

```
## [1] 0.25
```

The summary shows that if we don't specify the options in `svm()`, the default setting is radial kernel with $C = 1$ and $\gamma = 0.25$. If you'd like to change this setting, use `kernel` to specify the kernel, use `degree` to specify d for polynomial kernel, use `gamma` to specify γ for radial kernel, use `cost` to specify C. For more details, read the help document.

```
# test with train data
pred = predict(model, iris[,-5])
# Check accuracy:
table(pred, iris[,5])
```

```
##
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          2        48
```

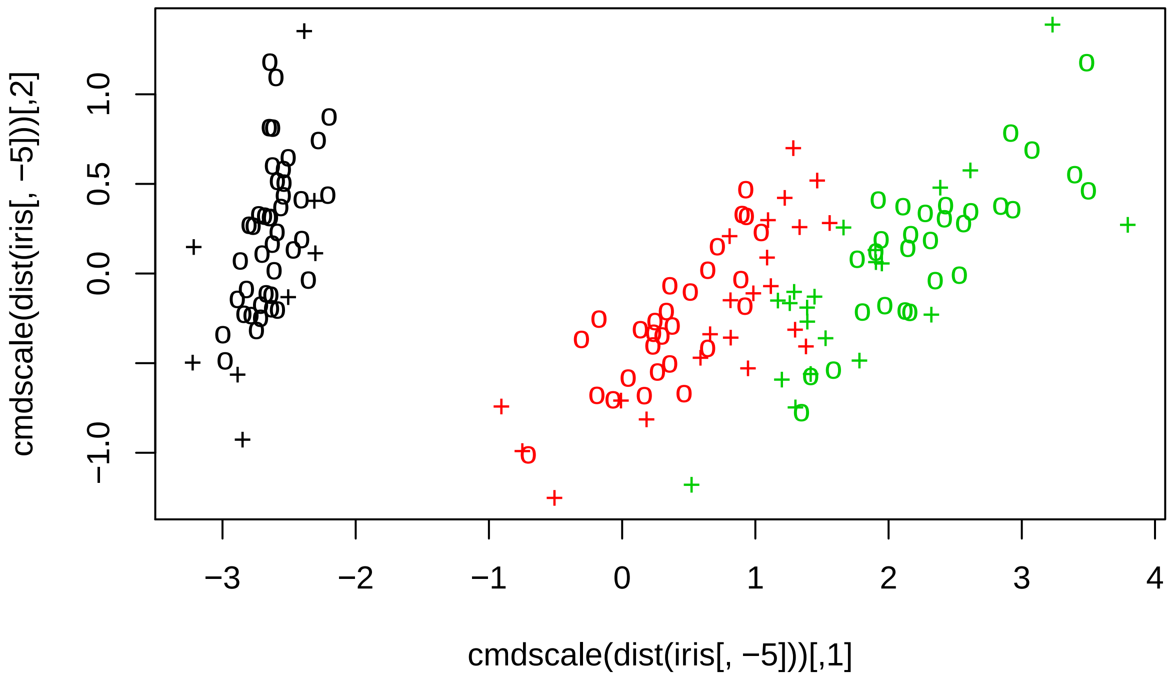
```
# compute decision values:
x = iris[, -5] #Check
pred = predict(model, x, decision.values = TRUE)
attr(pred, "decision.values")[1:4,]
```

```
##   setosa/versicolor setosa/virginica versicolor/virginica
## 1          1.196152         1.091757            0.6708810
## 2          1.064621         1.056185            0.8483518
## 3          1.180842         1.074542            0.6439798
## 4          1.110699         1.053012            0.6782041
```

```
pred[1:4]
```

```
##      1      2      3      4
## setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
# visualize (classes by color, SV by crosses):
plot(cmdscale(dist(iris[,-5])),
     col = as.integer(iris[,5]),
     pch = c("o","+")[1:150 %in% model$index + 1])
```



We can obtain the decision value by adding `decision.values = TRUE`. We can also visualise the training data where support vectors are labelled with crosses.

If you want to get the predicted probabilities, you have to enable the `probability = TRUE` option when train the model.

```
model = svm(iris[,-5], iris[,5], probability = TRUE)
pred = predict(model, iris[,-5], decision.values = TRUE, probability = TRUE)
attr(pred, "decision.values")[1:4,]
```

```
##   setosa/versicolor setosa/virginica versicolor/virginica
## 1          1.196152         1.091757            0.6708810
## 2          1.064621         1.056185            0.8483518
## 3          1.180842         1.074542            0.6439798
## 4          1.110699         1.053012            0.6782041
```

```
attr(pred, "probabilities")[1:4,]
```

```
##      setosa versicolor   virginica
## 1 0.9807210 0.01066424 0.008614765
## 2 0.9735828 0.01711589 0.009301278
## 3 0.9794506 0.01126504 0.009284338
## 4 0.9755471 0.01448395 0.009968988
```

2 Tune the parameters by cross-validatiaon

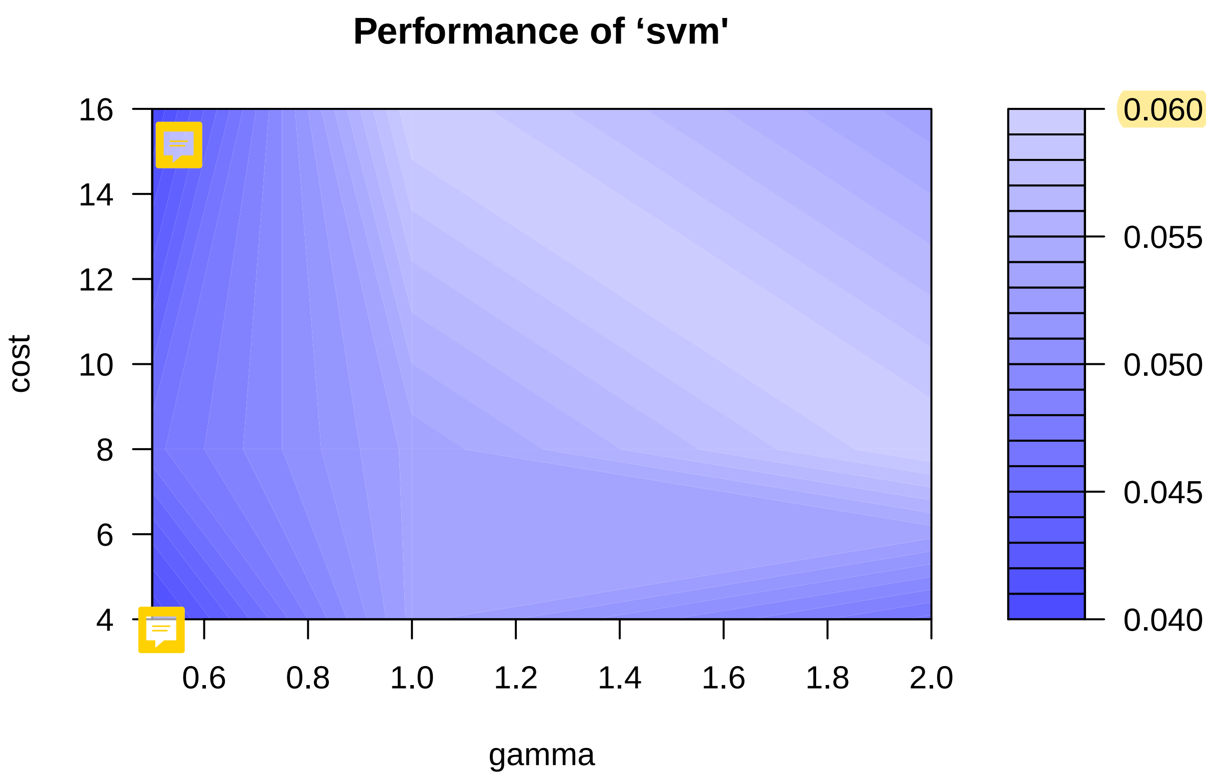
The package also provides you functions to tune the parameters. The following codes set the range of parameters C and γ in a list and tune the parameters based on 10-fold cross-validation. The plot shows you the changes of error rate with different C and γ. We can also use the best model (with smallest error rate) to predict instances.

```
set.seed(392)
tune.out = tune(svm, iris[,-5], iris[,5],
```

```
             ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
             tunecontrol = tune.control(sampling = "cross"),cross=10)
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    0.5    4
##
## - best performance: 0.04
##
## - Detailed performance results:
##   gamma cost      error dispersion
## 1   0.5    4 0.04000000 0.04661373
## 2   1.0    4 0.05333333 0.05258738
## 3   2.0    4 0.04666667 0.05488484
## 4   0.5    8 0.04666667 0.04499657
## 5   1.0    8 0.05333333 0.05258738
## 6   2.0    8 0.06000000 0.05837300
## 7   0.5   16 0.04000000 0.04661373
## 8   1.0   16 0.06000000 0.04919099
## 9   2.0   16 0.05333333 0.05258738
```

```
plot(tune.out)
```



```
# use the best model to predict the training instances
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = iris[, -5], train.y = iris[,
##     5], ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)), tunecontrol = tune.control(sampling = "cros
##     cross = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  4
##
## Number of Support Vectors:  49
```

```
pred=predict(tune.out$best.model,iris[,-5])
table(pred, iris[,5])
```

```
##
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         1
##   virginica       0          2        49
```

3 Use svm in caret

```
library(caret)
```

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

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

```
#load german credit data from caret package
data(GermanCredit)
# classify two status: good or bad
## Show the first 10 columns
str(GermanCredit[, 1:10])
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

```
## Delete two variables where all values are the same for both classes
GermanCredit[,c("Purpose.Vacation","Personal.Female.Single")] <- list(NULL)
#Get training and test sets
```

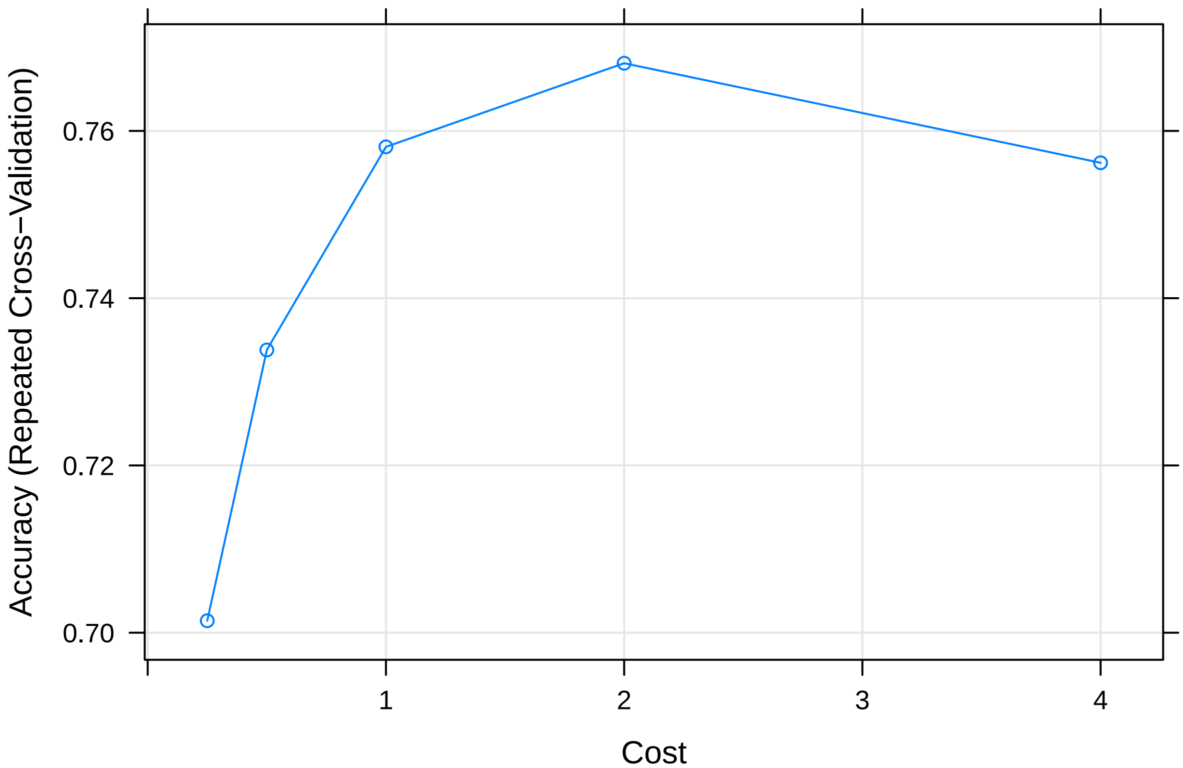
```r
set.seed(12)
trainIndex = createDataPartition(GermanCredit$Class, p = 0.7, list = FALSE, times = 1)
train=GermanCredit[trainIndex,] # training set
test=GermanCredit[-trainIndex,-10] # test set
```

Suppose we would like to use the RBF kernel.

```r
fitControl=trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 3)
set.seed(2333)
svm.Radial=train(Class ~., data = train, method = "svmRadial",
                 trControl=fitControl,
                 preProcess = c("center", "scale"),
                 tuneLength = 5)
svm.Radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 700 samples
##  59 predictor
##   2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 560, 560, 560, 560, 560, 560, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7014286  0.006603774
##   0.50  0.7338095  0.171407126
##   1.00  0.7580952  0.323141693
##   2.00  0.7680952  0.389344389
##   4.00  0.7561905  0.374952840
##
## Tuning parameter 'sigma' was held constant at a value of 0.01041683
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01041683 and C = 2.
```

```r
plot(svm.Radial)
```



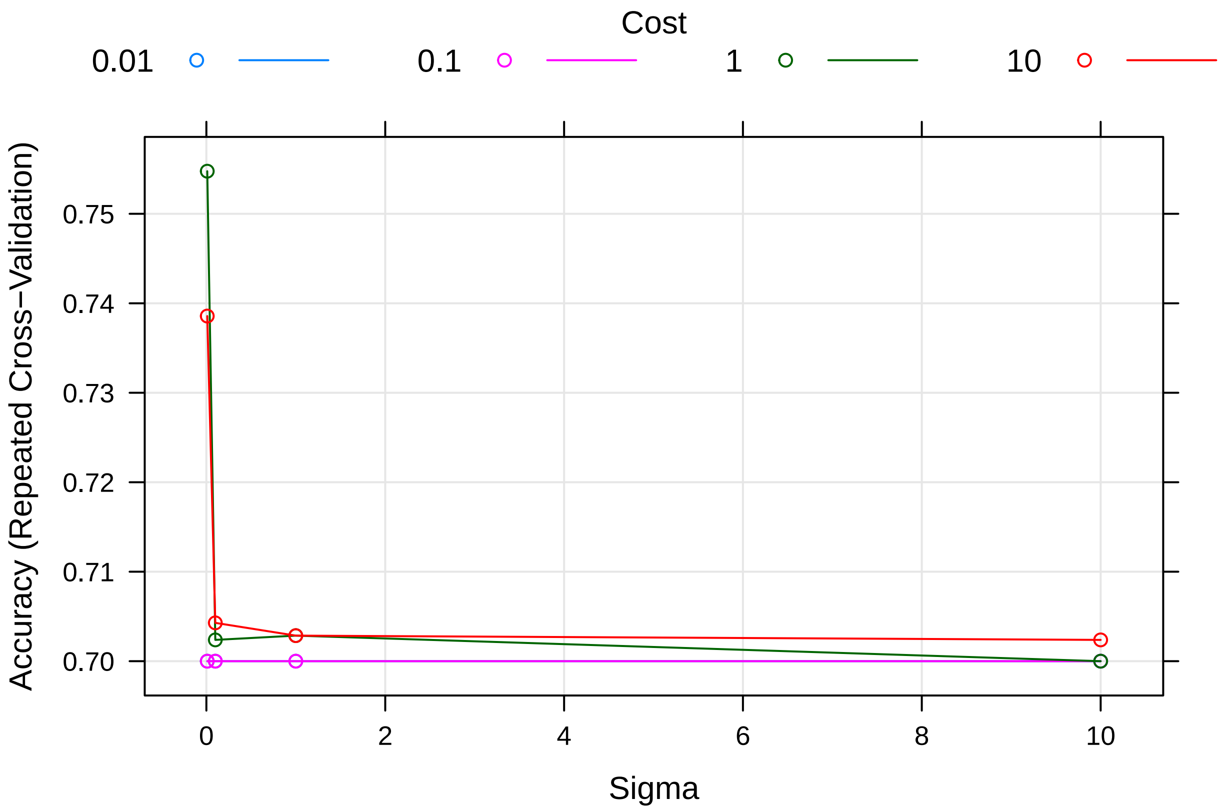
If we would like to tune both parameters, C and sigma.

```
grid_radial=expand.grid(sigma = c(0.01, 0.1, 1,10),
                             C = c(0.01, 0.1, 1, 10))
fitControl=trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 3)
set.seed(2333)
svm.Radialg=train(Class ~., data = train, method = "svmRadial",
                           trControl=fitControl,
                           preProcess = c("center", "scale"),
                           tuneGrid = grid_radial)
svm.Radialg
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 700 samples
##  59 predictor
##   2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 560, 560, 560, 560, 560, 560, ...
## Resampling results across tuning parameters:
##
##   sigma  C      Accuracy   Kappa
##    0.01   0.01  0.7000000  0.00000000
##    0.01   0.10  0.7000000  0.00000000
##    0.01   1.00  0.7547619  0.31277550
```

```
##     0.01  10.00  0.7385714  0.35455922
##     0.10   0.01  0.7000000  0.00000000
##     0.10   0.10  0.7000000  0.00000000
##     0.10   1.00  0.7023810  0.01351026
##     0.10  10.00  0.7042857  0.04019429
##     1.00   0.01  0.7000000  0.00000000
##     1.00   0.10  0.7000000  0.00000000
##     1.00   1.00  0.7028571  0.01320755
##     1.00  10.00  0.7028571  0.01320755
##    10.00   0.01  0.7000000  0.00000000
##    10.00   0.10  0.7000000  0.00000000
##    10.00   1.00  0.7000000  0.00000000
##    10.00  10.00  0.7023810  0.01100629
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 1.
```

```r
plot(svm.Radialg)
```



4 Visualise SVM

Visualise the effect of C for linear SVM.

```r
library(shiny)
library(shinythemes)
library(e1071)
# Define UI for app that draws a histogram ----
```

```r
ui <- fluidPage(
  # App title ----
  titlePanel("The effect of C in linear SVM"),
  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "C",
                  label = "Value of C in Lagrangian formulation:",
                  min = 1,
                  max = 50,
                  value = 30,
                  step=1)
    ),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: scatter plot ----
      plotOutput(outputId = "svmPlot")
    )
  )
)
# Define server logic required to draw a histogram ----
server <- function(input, output) {
  output$svmPlot <- renderPlot({
    ##### load training data
    set.seed(10111)
    x = matrix(rnorm(40), 20, 2)
    y = rep(c(-1, 1), c(10, 10))
    x[y == 1, ] = x[y == 1, ] + 1
    ##### fit linear SVM
    library(e1071)
    dat = data.frame(x, y = as.factor(y))
    c=input$C
    svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = c, scale = FALSE)
    ##### plot support vectors
    make.grid = function(x, n = 75) {
      grange = apply(x, 2, range)
      x1 = seq(from = grange[1, 1], to = grange[2, 1], length = n)
      x2 = seq(from = grange[1, 2], to = grange[2, 2], length = n)
      expand.grid(X1 = x1, X2 = x2)
    }
    xgrid = make.grid(x)
    ygrid = predict(svmfit, xgrid)
    ##### plot margin and classification boundary
    beta = drop(t(svmfit$coefs) %*% x[svmfit$index, ])
    beta0 = svmfit$rho
    plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
    points(x, col = y + 3, pch = 19)
    points(x[svmfit$index, ], pch = 5, cex = 2)
    abline(beta0/beta[2], -beta[1]/beta[2])
    abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
    abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
```

```
  })
}
shinyApp(ui = ui, server = server)
```

Visualise SVM with different kernels.

```
library(shiny)
library(shinythemes)
library(e1071)
# Define UI for app that draws a histogram ----
ui <- fluidPage(
  # App title ----
  titlePanel("The effect of kernels in SVM"),
  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      selectInput(inputId = "kernel", label = strong("Kernel"),
                  choices = c("linear","polynomial","RBF"),
                  selected = "linear"),
      sliderInput(inputId = "C",
                  label = "Value of C in Lagrangian formulation:",
                  min = 1,
                  max = 50,
                  value = 30,
                  step=1),
      sliderInput(inputId = "gamma",
                  label = "Value of Gamma:",
                  min = 0.01,
                  max = 1,
                  value = 30,
                  step=0.02),
      sliderInput(inputId = "degree",
                  label = "Value of Degree:",
                  min = 1,
                  max = 6,
                  value = 30,
                  step=1)
    ),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: scatter plot ----
      p("Linear: C"),
      p("Polynomial: C, degree, gamma"),
      p("RBF: C, gamma"),
      plotOutput(outputId = "svmPlot")
    )
  )
)
# Define server logic required to draw a projection plot ----
server <- function(input, output) {
  output$svmPlot <- renderPlot({
    ### load data
    load(url("http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/ESL.mixture.rda"))
```

```
    dat = data.frame(y = factor(ESL.mixture$y), ESL.mixture$x)
    ### fit svm
    k=input$kernel
    c=input$C; gamma=input$gamma; degree=input$degree
    if(k=="linear"){
      fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost = c)
    }else if(k=="polynomial"){
      fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "poly", degree=degree, gamma=gamma, 
    }else{
      fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "radial", cost = c, gamma=gamma)
    }
    ### plot classification boundary
    px1=ESL.mixture$px1; px2=ESL.mixture$px2
    x=ESL.mixture$x; y=ESL.mixture$y
    xgrid = expand.grid(X1 = px1, X2 = px2)
    ygrid = predict(fit, xgrid)
    plot(xgrid, col = as.numeric(ygrid), pch = 20, cex = 0.2)
    points(x, col = y + 1, pch = 19)
    func = predict(fit, xgrid, decision.values = TRUE)
    func = attributes(func)$decision
    xgrid = expand.grid(X1 = px1, X2 = px2)
    ygrid = predict(fit, xgrid)
    plot(xgrid, col = as.numeric(ygrid), pch = 20, cex = 0.2)
    points(x, col = y + 1, pch = 19)
    contour(px1, px2, matrix(func, 69, 99), level = 0, add = TRUE)
  })
}
shinyApp(ui = ui, server = server)
```