Math574M_Hw6

Saheed Adisa, Ganiyu

2023-11-15

(Elastic Net) Fit the elastic net for the prostate cancer data set using R package glmnet. Code example is given by elastic-net.R, posted at D2L lecture notes. The penalty term used in the package is

$$\lambda \left(\frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right),$$

where $\alpha \in [0,1]$ is a mixing parameter. It is the lasso penalty when $\alpha = 1$ and ridge penalty when $\alpha = 0$.

(a) Set the random seed to 2345. Consider five choices of $\alpha : 0, 0.25, 0.5, 0.75, 1$. For each α, fit the elastic net model using the training set and tune λ by 5 -fold CV using the minimum CV rule. For each α, report the best λ, its CV error, and the estimated regression coefficients.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
set.seed(2345)

# loading the dataset from the book site
prostate_data <- read.table(
  "https://hastie.su.domains/ElemStatLearn/datasets/prostate.data")
x_train <- subset(prostate_data, train == TRUE)[,1:9]
y_train <- subset(prostate_data, train == TRUE)[,9]
x_test <- subset(prostate_data, train == FALSE)[,1:9]
y_test <- subset(prostate_data, train == FALSE)[,9]
X <- as.matrix(x_train[,-9])
y <- y_train

alpha_values <- c(0, 0.25, 0.5, 0.75, 1)
best_lambda <- numeric(length(alpha_values))
cv_errors <- numeric(length(alpha_values))
coefficients_list <- list()
```

```r
for (i in seq_along(alpha_values)) {
  alpha <- alpha_values[i]

  # Fit elastic net model with cross-validated lambda selection
  cv_model <- cv.glmnet(X, y, alpha = alpha, nfolds = 5)

  # Get the best lambda and its corresponding CV error
  best_lambda[i] <- cv_model$lambda.min
  cv_errors[i] <- min(cv_model$cvm)

  # Get the estimated coefficients for the best lambda
  coefficients_list[[i]] <- coef(cv_model, s = best_lambda[i])

  # Display results for each alpha
  cat("\nAlpha =", alpha, "\n")
  cat("Best Lambda =", best_lambda[i], "\n")
  cat("CV Error =", cv_errors[i], "\n")
  print(coefficients_list[[i]])
  cat("========================================\n")
}
```

```
##
## Alpha = 0
## Best Lambda = 0.08788804
## CV Error = 0.6109575
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.095549713
## lcavol       0.492656404
## lweight      0.601227708
## age         -0.014818243
## lbph         0.137965816
## svi          0.679288020
## lcp         -0.116652810
## gleason      0.017256035
## pgg45        0.007077847
## ========================================
##
## Alpha = 0.25
## Best Lambda = 0.04041996
## CV Error = 0.5467462
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.165253942
## lcavol       0.521684733
## lweight      0.594630605
## age         -0.014354528
## lbph         0.135292199
## svi          0.665661192
## lcp         -0.127502524
## gleason      .
## pgg45        0.007197356
## ========================================
```

```
##
## Alpha = 0.5
## Best Lambda = 0.01841458
## CV Error = 0.5926155
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.186497208
## lcavol       0.543673205
## lweight      0.600931946
## age         -0.015827843
## lbph         0.137269548
## svi          0.684797626
## lcp         -0.152965906
## gleason      .
## pgg45        0.007674636
## ========================================
##
## Alpha = 0.75
## Best Lambda = 0.001585556
## CV Error = 0.5875752
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.372728518
## lcavol       0.572661116
## lweight      0.613298204
## age         -0.018702046
## lbph         0.143844962
## svi          0.731483353
## lcp         -0.199911877
## gleason     -0.021269500
## pgg45        0.009164638
## ========================================
##
## Alpha = 1
## Best Lambda = 0.003985616
## CV Error = 0.5997852
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.260075161
## lcavol       0.565509745
## lweight      0.611306691
## age         -0.018057803
## lbph         0.141639332
## svi          0.719657016
## lcp         -0.187508513
## gleason     -0.005108384
## pgg45        0.008564374
## ========================================
```

(b) Based on the reported CV errors in (a), select the best (α, λ) pair and report its TestErr.

```r
# getting best alpha and lambda
best_alpha_index <- which.min(cv_errors)
best_alpha <- alpha_values[best_alpha_index]
best_lambda_for_alpha <- best_lambda[best_alpha_index]
test_x <- as.matrix(x_test[,-9])
test_y <- y_test

# Fit the final elastic net model with the best alpha and lambda
final_model <- glmnet(X, y, alpha = best_alpha, lambda = best_lambda_for_alpha)

# Predict on the test set
test_predictions <- predict(final_model, newx = as.matrix(test_x), s = best_lambda_for_alpha)

# Calculate TestErr
test_error <- mean((test_predictions - test_y)^2)

cat("\nBest Alpha =", best_alpha, "\n")
```

```
##
## Best Alpha = 0.25
```

```r
cat("Best Lambda =", best_lambda_for_alpha, "\n")
```

```
## Best Lambda = 0.04041996
```

```r
cat("TestErr =", test_error, "\n")
```

```
## TestErr = 0.4919213
```

6. (Regression Splines) Consider one-dimensional regression model

$$Y = f(x) + \epsilon,$$

where the input $X \in [0, 1]$, the output $Y \in R$, and the underlying true function is

$$f(x) = 3\left[0.1\sin(2\pi x) + 0.2\cos(2\pi x) + 0.3\sin^2(2\pi x) + 0.4\cos^3(2\pi x) + 0.5\sin^3(2\pi x)\right]$$

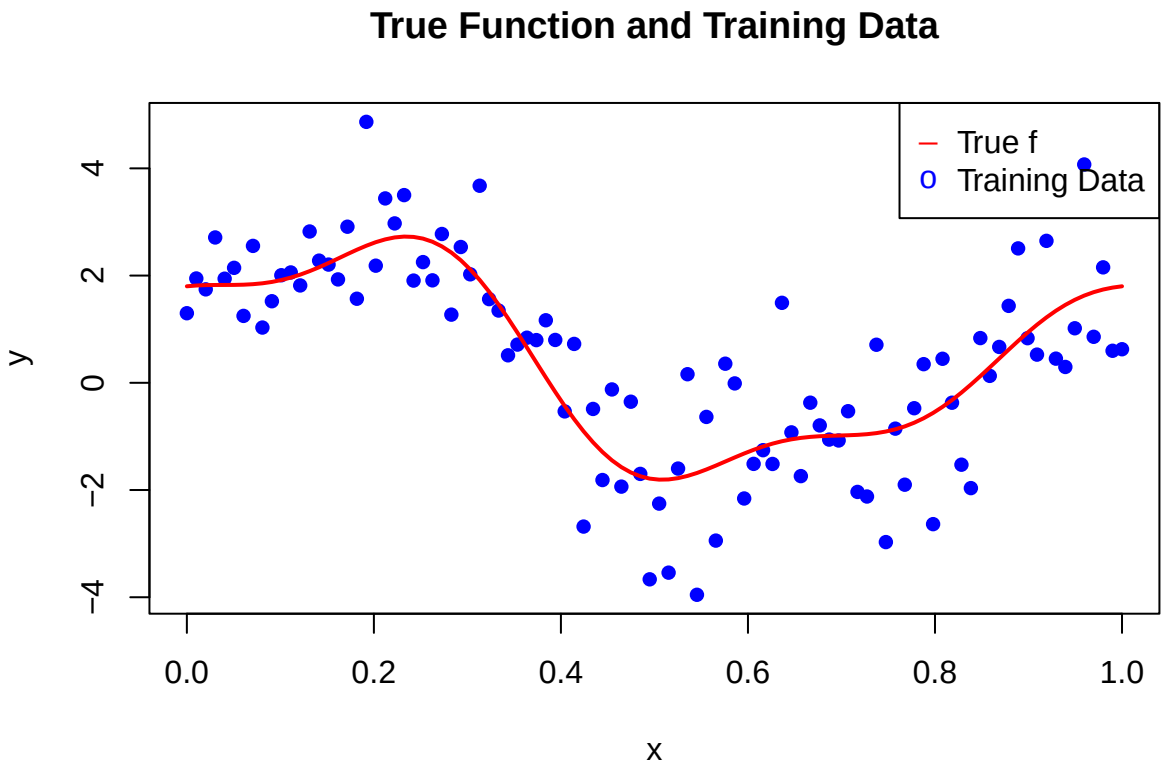
and the random error $\epsilon \sim N(0, 1)$ i.i.d.

(a) Set the random seed to 100 . Generate the training set $\{(x_i, y_i), i = 1, \cdots, n = 100\}$ as follows

$$\mathrm{x} = \mathrm{seq}(0, 1,\ \mathrm{length}\ = 100), \quad \mathrm{noise}\ = \mathrm{rnorm}(\ \mathrm{length}\ (\mathrm{x}), 0, 1), \quad \mathrm{y} = \mathrm{f}(\mathrm{x}) +\ \mathrm{noise}\ .$$

```r
set.seed(100)

# Define the true function f(x)
f <- function(x) {
  3 * (0.1 * sin(2 * pi * x) + 0.2 * cos(2 * pi * x) + 0.3 * sin(2 * pi * x)^2 +
```

```r
      0.4 * cos(2 * pi * x)^3 + 0.5 * sin(2 * pi * x)^3)
}

# Generate training data
n <- 100
x <- seq(0, 1, length = n)
noise <- rnorm(length(x), 0, 1)
y <- f(x) + noise

# Plot true function and training data
plot(x, y, col = "blue", pch = 16, main = "True Function and Training Data")
lines(x, f(x), col = "red", lwd = 2)  # True function
legend("topright", legend = c("True f", "Training Data"), col = c("red", "blue"), pch = c("_", "o"))
```



(b) Estimate f using four methods: linear model, quadratic model, cubic model, and cubic splines with $\mathrm{df} = 9$. For each model, obtain $\hat{f}$ from the training data and compute its mean squared error using

$$\mathrm{MSE}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left[f(x_i) - \hat{f}(x_i)\right]^2.$$

```r
# Linear model
linear_model <- lm(y ~ x)
linear_pred <- predict(linear_model, data.frame(x = x))
mse_linear <- mean((f(x) - linear_pred)^2)

# Quadratic model
```

```r
quadratic_model <- lm(y ~ poly(x, 2))
quadratic_pred <- predict(quadratic_model, data.frame(x = x))
mse_quadratic <- mean((f(x) - quadratic_pred)^2)

# Cubic model
cubic_model <- lm(y ~ poly(x, 3))
cubic_pred <- predict(cubic_model, data.frame(x = x))
mse_cubic <- mean((f(x) - cubic_pred)^2)

# Cubic splines with df=9
library(splines)
cubic_spline_model <- lm(y ~ bs(x, df = 9))
cubic_spline_pred <- predict(cubic_spline_model, data.frame(x = x))
mse_cubic_spline <- mean((f(x) - cubic_spline_pred)^2)
```

(c) Compare the four estimated $\hat{f}$ in terms of MSE and comment on their performance. Create one plot which contains the scatter plot (x-axis is x, y-axis is y) of the training data, the true f, superposed with the four estimated $\hat{f}$, using different colors and adding a legend.

```r
models <- c("Linear", "Quadratic", "Cubic", "Cubic Spline (df=9)")
mses <- c(mse_linear, mse_quadratic, mse_cubic, mse_cubic_spline)
cat("Model\t\tMSE\n")
```

```
## Model        MSE
```

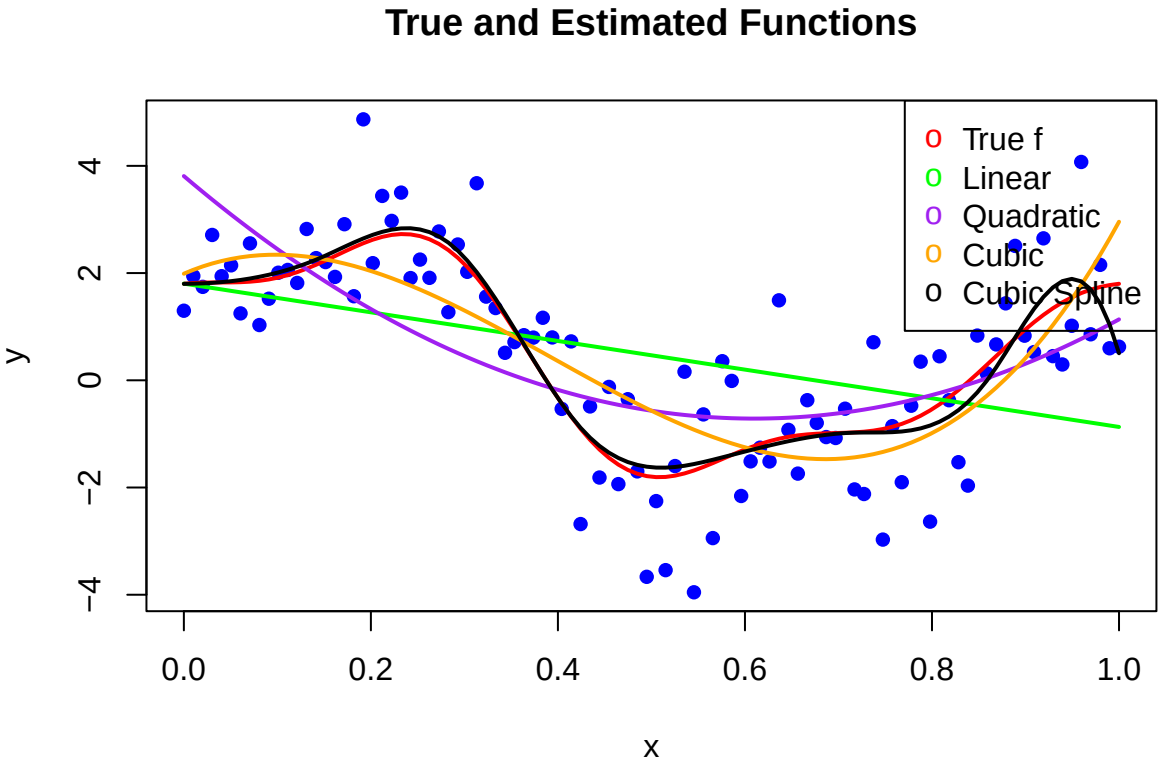
```r
cat("-----\t\t---\n")
```

```
## -----        ---
```

```r
for (i in seq_along(models)) {
  cat(sprintf("%s:\t%.4f\n", models[i], mses[i]))
}
```

```
## Linear:  1.8639
## Quadratic:   0.9308
## Cubic:   0.4191
## Cubic Spline (df=9): 0.0453
```

```r
# Plot true function and estimated functions
plot(x, y, col = "blue", pch = 16, main = "True and Estimated Functions")
lines(x, f(x), col = "red", lwd = 2)  # True function
lines(x, linear_pred, col = "green", lwd = 2)  # Linear model
lines(x, quadratic_pred, col = "purple", lwd = 2)  # Quadratic model
lines(x, cubic_pred, col = "orange", lwd = 2)  # Cubic model
lines(x, cubic_spline_pred, col = "black", lwd = 2)  # Cubic spline
legend("topright", legend = c("True f", "Linear", "Quadratic", "Cubic", "Cubic Spline"),
       col = c("red", "green", "purple", "orange", "black"), pch = c("o", "o", "o", "o", "o"))
```



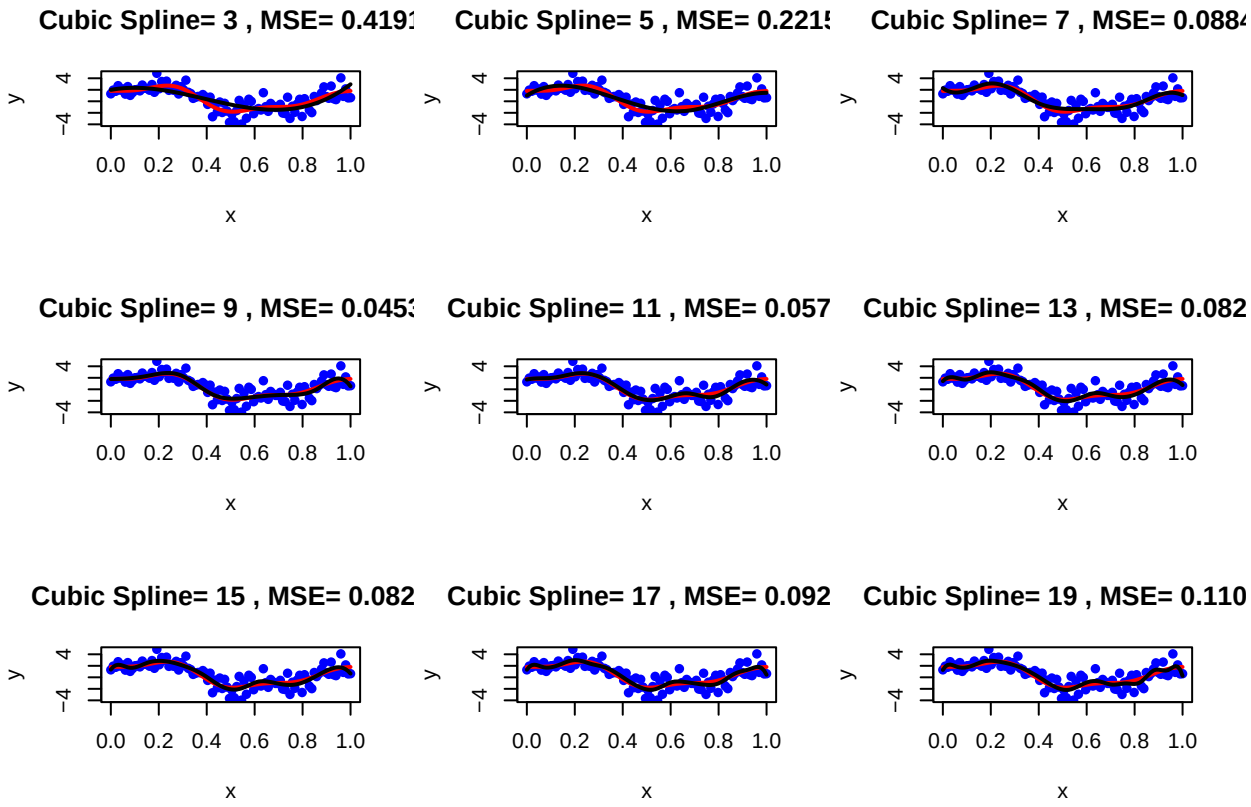
Comment: We notice that out of the four methods, Cubic Spline with $df = 9$ performed best with 0.0453 error, while Linear model performed worst with 1.8639 error. We can also see from the plotted graph how Cubic Sline curve is much closed to true function, while Linear line is bias.

(d) Estimate f by regression cubic splines with nine values of $\mathrm{df} \in \{3, 5, \cdots, 19\}$. For each df, obtain $\hat{f}$, compute its mean squared error $\mathrm{MSE}\left(\hat{f}_{df}\right)$, and draw a scatter plot of the training data superposed by the true f and $\hat{f}_{df}$; organize these nine plots into one figure with 3×3 display using par(mfrow).

```
# Regression cubic splines with different values of df
par(mfrow = c(3, 3))
for (df in seq(3, 19, by = 2)) {
  cubic_spline_model <- lm(y ~ bs(x, df = df))
  cubic_spline_pred <- predict(cubic_spline_model, data.frame(x = x))
  mse_cubic_spline <- mean((f(x) - cubic_spline_pred)^2)

  # Plot true function and estimated function for each df
  plot(x, y, col = "blue", pch = 16,
       main = paste("Cubic Spline=", df, ", MSE=",round(mse_cubic_spline,4)))
  lines(x, f(x), col = "red", lwd = 2)  # True function
  lines(x, cubic_spline_pred, col = "black", lwd = 2)  # Cubic spline
  #legend("topright", legend = c("True f", "Cubic Spline"),
         # col = c("red", "black"), pch = c("o", "o"))
  cat(sprintf("DF=%d:\t%.4f\n", df, mse_cubic_spline))
}
```

```
## DF=3:    0.4191
```

```
## DF=5:    0.2215

## DF=7:    0.0884

## DF=9:    0.0453

## DF=11:   0.0571

## DF=13:   0.0823

## DF=15:   0.0826

## DF=17:   0.0927
```




```
## DF=19:   0.1104
```

(e) Based on (d), which df gives the best performance? Justify your answer.

Comment: With 9 different df on the cubic spline, we have cubic spline with df=9 performed best, which its curve is also the most closed to the true function, where the one of df=11 is the second best. Then, cubic spline with df=3 gives the lowest performance. Therefore, we observed that increasing the degree of freedom (df) doesn't guarantee better performance, which may lead to overfitting, that is, high variance, low bias.

7.(Smoothing Splines) Consider one-dimensional regression model with the input $X \in [0,1]$ and the output Y. Assume the true regression model is

$$Y = f(x) + \epsilon,$$

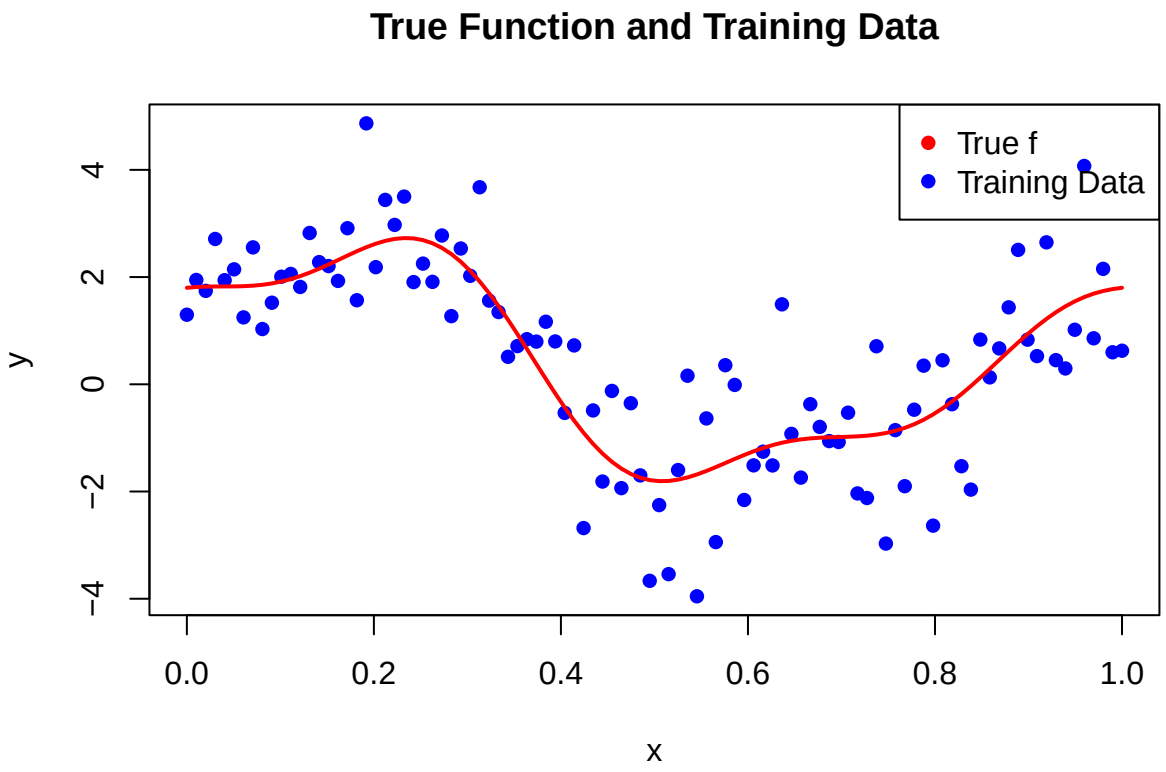
with the true f given by

$$f(x) = 3\left[0.1\sin(2\pi x) + 0.2\cos(2\pi x) + 0.3\sin^2(2\pi x) + 0.4\cos^3(2\pi x) + 0.5\sin^3(2\pi x)\right]$$

and the random error $\epsilon \sim N(0,1)$ i.i.d. See the sample code in splineCode.R.\

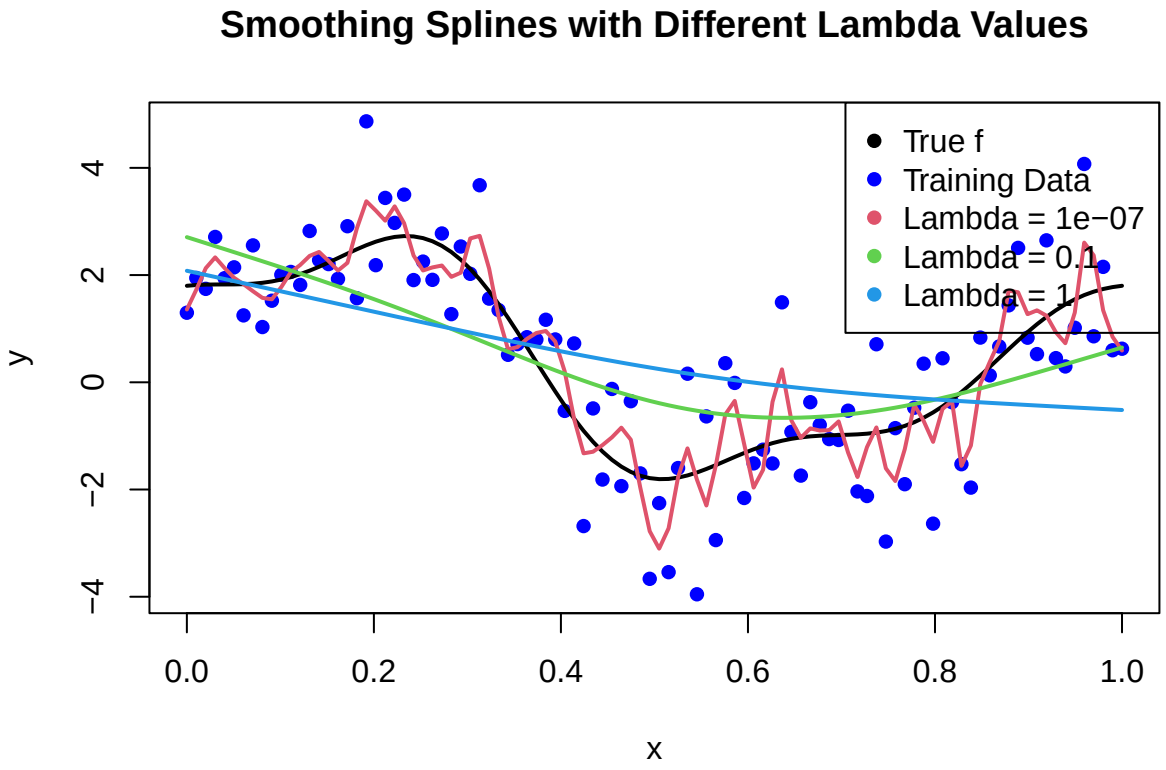
(a) Generate the training data using the procedure described in 7(a), with $n = 100$.

```r
set.seed(100)
library(splines)

# The true function f(x)
f <- function(x) {
  3 * (0.1 * sin(2 * pi * x) + 0.2 * cos(2 * pi * x) + 0.3 * sin(2 * pi * x)^2 +
       0.4 * cos(2 * pi * x)^3 + 0.5 * sin(2 * pi * x)^3)
}

# Generate training data
n <- 100
x <- seq(0, 1, length = n)
noise <- rnorm(length(x), 0, 1)
y <- f(x) + noise

# Plot true function and training data
plot(x, y, col = "blue", pch = 16, main = "True Function and Training Data")
lines(x, f(x), col = "red", lwd = 2)
legend("topright", legend = c("True f", "Training Data"), col = c("red", "blue"), pch = c(16, 16))
```



(b) Estimate f by fitting smoothing splines with three values of $\lambda \in \{10^{-7}, 10^{-1}, 1\}$, and compute the $\mathrm{MSE}\left(\hat{f}_\lambda\right)$. Plot the training data scatter plot, superposed by the true f and three fitted curves.

```r
lambda_values <- c(1e-7, 1e-1, 1)
mse_values <- numeric(length(lambda_values))

# Plot training data and true function
plot(x, y, col = "blue", pch = 16, main = "Smoothing Splines with Different Lambda Values")
lines(x, f(x), col = "black", lwd = 2)

# Fit smoothing splines with different lambda values
for (i in seq_along(lambda_values)) {
  lambda <- lambda_values[i]
  spline_model <- smooth.spline(x, y, lambda = lambda)
  y_pred <- predict(spline_model, x)$y
  mse_values[i] <- mean((f(x) - y_pred)^2)

  # Plot fitted curve
  lines(x, y_pred, col = i + 1, lwd = 2)
}
legend("topright", legend = c("True f", "Training Data", paste("Lambda =", lambda_values)),
       col = c("black", "blue", 2:(length(lambda_values) + 1)), pch = c(16, 16, 16))
```




```
# Print MSE values
cat("Lambda\t\tMSE\n")
```

```
## Lambda       MSE
```

```
cat("------\t\t---\n")
```

```
## ------       ---
```

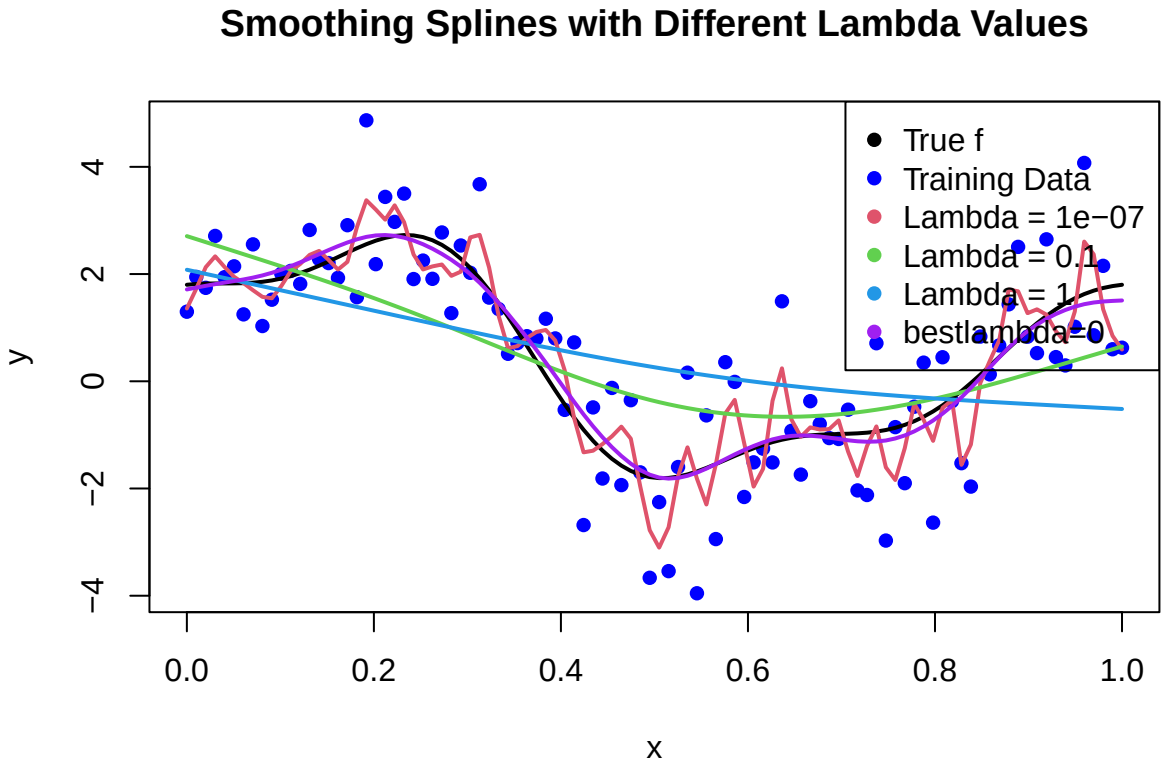
```
for (i in seq_along(lambda_values)) {
  cat(sprintf("%.7f:\t%.4f\n", lambda_values[i], mse_values[i]))
}
```

```
## 0.0000001:   0.3170
## 0.1000000:   0.7516
## 1.0000000:   1.5302
```

(c) Find the best $\hat{\lambda}$ using GCV, compute its $\mathrm{MSE}\left(\hat{f}_{\hat{\lambda}}\right)$, and compare it with those in (b). Add the fitted curve $\hat{f}_{\hat{\lambda}}$ to the figure produced in (b), including a legend.

```
# Plot training data and true function
plot(x, y, col = "blue", pch = 16, main = "Smoothing Splines with Different Lambda Values")
lines(x, f(x), col = "black", lwd = 2)

# Fit smoothing splines with different lambda values
```

```r
for (i in seq_along(lambda_values)) {
  lambda <- lambda_values[i]
  spline_model <- smooth.spline(x, y, lambda = lambda)
  y_pred <- predict(spline_model, x)$y
  mse_values[i] <- mean((f(x) - y_pred)^2)

  # Plot fitted curve
  lines(x, y_pred, col = i + 1, lwd = 2)
}
# use GCV to tune parameter
sfit <- smooth.spline(x,y)
bestlam <- round(sfit$lambda,3)
lines(predict(sfit,x), col = "purple", lty=1, lwd=2)
besterr <- mean((predict(sfit,x)$y-f(x))^2)
legend("topright", legend = c("True f", "Training Data", paste("Lambda =", lambda_values), "bestlambda=
       col = c("black", "blue", 2:(length(lambda_values) + 1), "purple"), pch = c(16, 16, 16,16))
```




```r
# Print MSE values
cat("Lambda\t\tMSE\n")
```

```
## Lambda       MSE
```

```r
cat("------\t\t---\n")
```

```
## ------       ---
```

```
for (i in seq_along(lambda_values)) {
  cat(sprintf("%.7f:\t%.4f\n", lambda_values[i], mse_values[i]))
}
```

```
## 0.0000001:   0.3170
## 0.1000000:   0.7516
## 1.0000000:   1.5302
```

```
cat(sprintf("%.7f:\t%.4f\n", bestlam, besterr))   # Adding best lambda MSE
```

```
## 0.0000000:   0.0208
```

Comment: We notice that the best lambda selected by GCV outperform others lambdas. Also, its curve in the plot is the most closest one to the true function even closer than the curve of Cubic Spline with df=9 in problem 6.

8. Classify 2's and 3's for the zip code set. Apply the SVM using the R package *e*1071.

(a) Fit the linear SVM using the training data with a sequence of tuning parameters. You can create a grid of points for the cost parameter, e.g. $\{10^{-3}, 10^{-2.5}, 10^{-2}, \cdots, 10^{4.5}, 10^{5}\}$**. Compute the test error for each cost parameter, and report the best one which gives the smallest test error.**

```
library(e1071)
library(caret)
```

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

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

```
set.seed(123)

# Loading the dataset
train_data <- read.table("zip.train", sep = "")
test_data <- read.table("zip.test", sep = "")

# taking the subsets of the data with class Y in {0, 3, 5, 6, 9} for part (a)
class_labels <- c(2, 3)
train_data <- train_data[train_data$V1 %in% class_labels, ]
test_data <- test_data[test_data$V1 %in% class_labels, ]

cost_values <- 10^(seq(-3, 5, by = 0.5))   # cost values

# Using 10-fold cross validation in order to determine optimal cost parameter
tuned1 <- tune(svm, V1 ~ ., data = train_data, kernel = "linear", ranges = list(cost=cost_values))
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.
```

```r
summary(tuned1)
```

```
##
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.03498357
##
## - Detailed performance results:
##            cost      error  dispersion
## 1  1.000000e-03 0.03498357 0.006712169
## 2  3.162278e-03 0.03634154 0.006898645
## 3  1.000000e-02 0.03798780 0.006865907
## 4  3.162278e-02 0.03976667 0.006882071
## 5  1.000000e-01 0.04123541 0.007219925
## 6  3.162278e-01 0.04205254 0.007588373
## 7  1.000000e+00 0.04204313 0.007575988
## 8  3.162278e+00 0.04242372 0.007598148
## 9  1.000000e+01 0.04741085 0.009569293
## 10 3.162278e+01 0.05531351 0.033583262
## 11 1.000000e+02 0.04924210 0.008858872
## 12 3.162278e+02 0.05525945 0.010501760
## 13 1.000000e+03 0.06561827 0.008938723
## 14 3.162278e+03 0.07866356 0.009966237
## 15 1.000000e+04 0.07875413 0.009812178
## 16 3.162278e+04 0.07875413 0.009812178
## 17 1.000000e+05 0.07875413 0.009812178
```

Checking for best model

```
bestmod1 = tuned1$best.model
summary ( bestmod1 )
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = V1 ~ ., data = train_data, ranges = list(cost = cost_values),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  0.001
##       gamma:  0.00390625
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1010
```

```r
cat("Best cost parameter for Linear SVM:", tuned1$best.parameters$cost, "\n")
```

```
## Best cost parameter for Linear SVM: 0.001
```

Plotting Confusion Matrix with its accuracy on test data for linear SVM

```r
x_test_pred1 <- predict(bestmod1, test_data, type="class")
test_error1 <- mean(round(x_test_pred1) != test_data$V1 )
cat("Test error=", test_error1, "\n")
```

```
## Test error= 0.03021978
```

```r
confusionMatrix(table(round(x_test_pred1), test_data$V1)[-3,])
```

```
## Confusion Matrix and Statistics
##
##
##       2   3
##   2 193   6
##   3   4 160
##
##                Accuracy : 0.9725
##                  95% CI : (0.9499, 0.9867)
##     No Information Rate : 0.5427
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9444
##
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.9797
##             Specificity : 0.9639
##          Pos Pred Value : 0.9698
##          Neg Pred Value : 0.9756
##              Prevalence : 0.5427
##          Detection Rate : 0.5317
##    Detection Prevalence : 0.5482
##       Balanced Accuracy : 0.9718
##
##        'Positive' Class : 2
##
```

(b) Fit the polynomial kernel SVM using the training data. In this case, there are two tuning parameters: cost and the polynomial degree, so you need to search for the best pair in a two-dimensional grid, say the cost in $\{10^{-3}, 10^{-2.5}, 10^{-2}, \cdots, 10^{4.5}, 10^{5}\}$ and the degree in $\{1, 2, 3\}$. Compute the test errors for all the possible pairs and identify the best pair that gives the smallest test error.

```r
# Using 10-fold cross validation in order to determine optimal degree, and cost parameter
tuned2 <- tune(svm, V1 ~ ., data = train_data, kernel = "polynomial", ranges = list(cost=cost_values, de
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.
```

```
summary(tuned2)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      3
##
## - best performance: 0.02901248
##
## - Detailed performance results:
##            cost degree      error  dispersion
## 1  1.000000e-03      1 0.30450667 0.035525858
## 2  3.162278e-03      1 0.13089186 0.021718061
## 3  1.000000e-02      1 0.05721391 0.007752731
## 4  3.162278e-02      1 0.04225198 0.006864186
## 5  1.000000e-01      1 0.03599921 0.006254626
## 6  3.162278e-01      1 0.03474802 0.006323153
## 7  1.000000e+00      1 0.03596554 0.006398467
## 8  3.162278e+00      1 0.03808078 0.006842103
## 9  1.000000e+01      1 0.03994400 0.006711883
## 10 3.162278e+01      1 0.04119236 0.006919172
## 11 1.000000e+02      1 0.04227688 0.007887778
## 12 3.162278e+02      1 0.04254672 0.008111243
## 13 1.000000e+03      1 0.04283447 0.008119317
## 14 3.162278e+03      1 0.08332130 0.116550471
## 15 1.000000e+04      1 0.05267390 0.022214992
## 16 3.162278e+04      1 0.05258924 0.014159948
## 17 1.000000e+05      1 0.06086061 0.016963040
## 18 1.000000e-03      2 0.40673318 0.054528464
## 19 3.162278e-03      2 0.37287345 0.086448559
## 20 1.000000e-02      2 0.30801408 0.091718441
## 21 3.162278e-02      2 0.18059244 0.054789859
## 22 1.000000e-01      2 0.06706963 0.014042676
## 23 3.162278e-01      2 0.03980347 0.009522053
## 24 1.000000e+00      2 0.03331041 0.009926956
## 25 3.162278e+00      2 0.03415489 0.011875510
## 26 1.000000e+01      2 0.03456410 0.012505715
## 27 3.162278e+01      2 0.03456410 0.012505715
## 28 1.000000e+02      2 0.03456410 0.012505715
## 29 3.162278e+02      2 0.03456410 0.012505715
## 30 1.000000e+03      2 0.03456410 0.012505715
## 31 3.162278e+03      2 0.03456410 0.012505715
## 32 1.000000e+04      2 0.03456410 0.012505715
## 33 3.162278e+04      2 0.03456410 0.012505715
## 34 1.000000e+05      2 0.03456410 0.012505715
## 35 1.000000e-03      3 0.40442270 0.045624362
## 36 3.162278e-03      3 0.36446271 0.061036488
## 37 1.000000e-02      3 0.27452755 0.073652833
## 38 3.162278e-02      3 0.14015298 0.035950313
## 39 1.000000e-01      3 0.07752652 0.016141815
## 40 3.162278e-01      3 0.04551904 0.008559926
## 41 1.000000e+00      3 0.03077553 0.006998297
## 42 3.162278e+00      3 0.02907515 0.007857204
## 43 1.000000e+01      3 0.02901248 0.007871057
## 44 3.162278e+01      3 0.02901248 0.007871057
```

```
## 45 1.000000e+02      3 0.02901248 0.007871057
## 46 3.162278e+02      3 0.02901248 0.007871057
## 47 1.000000e+03      3 0.02901248 0.007871057
## 48 3.162278e+03      3 0.02901248 0.007871057
## 49 1.000000e+04      3 0.02901248 0.007871057
## 50 3.162278e+04      3 0.02901248 0.007871057
## 51 1.000000e+05      3 0.02901248 0.007871057
```

```
cat("Best degree and Cost Parameter for polynomial SVM:", tuned2$best.parameters$degree, ", and", tuned
```

```
## Best degree and Cost Parameter for polynomial SVM: 3 , and 10  respectively
```

Checking for best model

```
bestmod2 = tuned2$best.model
summary ( bestmod2 )
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = V1 ~ ., data = train_data, ranges = list(cost = cost_values,
##     degree = c(1, 2, 3)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.00390625
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  967
```

```
cat("Best cost and gamma parameter for polynomial SVM are:", tuned2$best.parameters$cost, "and", tuned2
```

```
## Best cost and gamma parameter for polynomial SVM are: 10 and 3  respectively.
```

Plotting Confusion Matrix with its accuracy on test data for polynomial SVM

```
x_test_pred2 <- predict(bestmod2, test_data, type="class")
test_error2 <- mean(round(x_test_pred2) != test_data$V1 )
cat("Test error=", test_error2, "\n")
```

```
## Test error= 0.02747253
```

```r
confusionMatrix(table(round(x_test_pred2), test_data$V1)[c(-1,-4),])
```

```
## Confusion Matrix and Statistics
##
##
##       2   3
##   2 193   4
##   3   4 161
##
##                Accuracy : 0.9779
##                  95% CI : (0.9569, 0.9904)
##     No Information Rate : 0.5442
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9555
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9797
##             Specificity : 0.9758
##          Pos Pred Value : 0.9797
##          Neg Pred Value : 0.9758
##              Prevalence : 0.5442
##          Detection Rate : 0.5331
##    Detection Prevalence : 0.5442
##       Balanced Accuracy : 0.9777
##
##        'Positive' Class : 2
##
```

(c) Fit the Gaussian kernel SVM using the training data with a sequence of tuning parameters. In addition to the cost parameter, the second tuning parameter is the relative bandwidth controlled by gamma in the function svm(). Therefore you should consider a combination of two, say the cost from $\{10^{-3}, 10^{-2.5}, 10^{-2}, \cdots, 10^{4.5}, 10^{5}\}$ and the gamma from $\{0.001, 0.005, 0.1\}$. Compute the test errors for all the pairs and identify the best pair that gives the smallest test error.

```r
# Using 10-fold cross validation in order to determine optimal gamma, and cost parameter
set.seed(10)
tuned3 <- tune(svm, V1 ~ ., data = train_data, kernel = "radial", ranges = list(cost=cost_values , gamma
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
```

```
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
```

```
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
```

```
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'V17' constant. Cannot scale data.
```

```r
summary(tuned3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10 0.001
##
## - best performance: 0.01829601
##
## - Detailed performance results:
##            cost gamma      error  dispersion
## 1  1.000000e-03 0.001 0.38230725 0.049955778
## 2  3.162278e-03 0.001 0.30049532 0.046449389
## 3  1.000000e-02 0.001 0.12726588 0.029340882
## 4  3.162278e-02 0.001 0.05484847 0.009466572
```

```
## 5  1.000000e-01 0.001 0.03837250 0.007257251
## 6  3.162278e-01 0.001 0.02900207 0.005171174
## 7  1.000000e+00 0.001 0.02362829 0.004187979
## 8  3.162278e+00 0.001 0.01966235 0.004129637
## 9  1.000000e+01 0.001 0.01829601 0.004379799
## 10 3.162278e+01 0.001 0.01864264 0.004422338
## 11 1.000000e+02 0.001 0.01880385 0.004374843
## 12 3.162278e+02 0.001 0.01880385 0.004374843
## 13 1.000000e+03 0.001 0.01880385 0.004374843
## 14 3.162278e+03 0.001 0.01880385 0.004374843
## 15 1.000000e+04 0.001 0.01880385 0.004374843
## 16 3.162278e+04 0.001 0.01880385 0.004374843
## 17 1.000000e+05 0.001 0.01880385 0.004374843
## 18 1.000000e-03 0.005 0.37614294 0.055180235
## 19 3.162278e-03 0.005 0.28763448 0.071843970
## 20 1.000000e-02 0.005 0.12593674 0.035609695
## 21 3.162278e-02 0.005 0.06423393 0.016890733
## 22 1.000000e-01 0.005 0.04092246 0.010759354
## 23 3.162278e-01 0.005 0.02766456 0.007197210
## 24 1.000000e+00 0.005 0.02315377 0.005480604
## 25 3.162278e+00 0.005 0.02301667 0.005297163
## 26 1.000000e+01 0.005 0.02300923 0.005277752
## 27 3.162278e+01 0.005 0.02300923 0.005277752
## 28 1.000000e+02 0.005 0.02300923 0.005277752
## 29 3.162278e+02 0.005 0.02300923 0.005277752
## 30 1.000000e+03 0.005 0.02300923 0.005277752
## 31 3.162278e+03 0.005 0.02300923 0.005277752
## 32 1.000000e+04 0.005 0.02300923 0.005277752
## 33 3.162278e+04 0.005 0.02300923 0.005277752
## 34 1.000000e+05 0.005 0.02300923 0.005277752
## 35 1.000000e-03 0.010 0.40086987 0.055900669
## 36 3.162278e-03 0.010 0.35533758 0.077135580
## 37 1.000000e-02 0.010 0.25895431 0.081330597
## 38 3.162278e-02 0.010 0.12056805 0.038219581
## 39 1.000000e-01 0.010 0.07735703 0.024749247
## 40 3.162278e-01 0.010 0.04920224 0.014557358
## 41 1.000000e+00 0.010 0.04002442 0.010107936
## 42 3.162278e+00 0.010 0.03999663 0.010008911
## 43 1.000000e+01 0.010 0.03999663 0.010008911
## 44 3.162278e+01 0.010 0.03999663 0.010008911
## 45 1.000000e+02 0.010 0.03999663 0.010008911
## 46 3.162278e+02 0.010 0.03999663 0.010008911
## 47 1.000000e+03 0.010 0.03999663 0.010008911
## 48 3.162278e+03 0.010 0.03999663 0.010008911
## 49 1.000000e+04 0.010 0.03999663 0.010008911
## 50 3.162278e+04 0.010 0.03999663 0.010008911
## 51 1.000000e+05 0.010 0.03999663 0.010008911
```

```
cat("Best cost and gamma parameter for Gaussian SVM are:", tuned3$best.parameters$cost, "and", tuned3$b
```

```
## Best cost and gamma parameter for Gaussian SVM are: 10 and 0.001  respectively.
```

Checking for best model

```
bestmod3 = tuned3$best.model
summary ( bestmod3 )
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = V1 ~ ., data = train_data, ranges = list(cost = cost_values,
##     gamma = c(0.001, 0.005, 0.01)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.001
##     epsilon:  0.1
##
##
## Number of Support Vectors:  820
```

```
cat("Best cost and gamma parameter for Gaussian SVM are:", tuned3$best.parameters$cost, "and", tuned3$be
```

```
## Best cost and gamma parameter for Gaussian SVM are: 10 and 0.001  respectively.
```

Plotting Confusion Matrix with its accuracy on test data for Gaussian SVM

```
x_test_pred3 <- predict(bestmod3, test_data, type="class")
test_error3 <- mean(round(x_test_pred3) != test_data$V1 )
cat("Test error=", test_error3, "\n")
```

```
## Test error= 0.02197802
```

```
confusionMatrix(table(round(x_test_pred3), test_data$V1))
```

```
## Confusion Matrix and Statistics
##
##
##       2   3
##   2 194   4
##   3   4 162
##
##                Accuracy : 0.978
##                  95% CI : (0.9572, 0.9905)
##     No Information Rate : 0.544
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9557
```

```
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9798
##             Specificity : 0.9759
##          Pos Pred Value : 0.9798
##          Neg Pred Value : 0.9759
##              Prevalence : 0.5440
##          Detection Rate : 0.5330
##    Detection Prevalence : 0.5440
##       Balanced Accuracy : 0.9779
##
##        'Positive' Class : 2
##
```

(d) Compare the best classifiers from (a)-(c) in terms of their test error performance and make comments.

```
cat("==================== Linear SVM ====================\n")
```

```
## ==================== Linear SVM ====================
```

```
cat("Test Error=",test_error1,",\t Best cost parameter", tuned1$best.parameters$cost, ".\n")
```

```
## Test Error= 0.03021978 ,  Best cost parameter 0.001 .
```

```
cat("\n==================== Polynomial SVM ====================\n")
```

```
##
## ==================== Polynomial SVM ====================
```

```
cat("Test Error=",test_error2,",\t Best cost and gamma parameter for polynomial SVM are:", tuned2$best.p
```

```
## Test Error= 0.02747253 ,  Best cost and gamma parameter for polynomial SVM are: 10 and 3  respectivel
```

```
cat("\n==================== Gaussian SVM ====================\n")
```

```
##
## ==================== Gaussian SVM ====================
```

```
cat("Test Error=",test_error3,",\t Best cost and gamma parameter for Gaussian SVM are:", tuned3$best.pa
```

```
## Test Error= 0.02197802 ,  Best cost and gamma parameter for Gaussian SVM are: 10 and 0.001  respecti
```

Comment: Based on their test errors, the Gaussian SVM performs best with 0.0219 test error, follow by Polynomial SVM with 0.0274 test error, and lowest performance goes to Linear SVM with 0.0302 test error.