STAT 4060J: Homework5

Boqian Wang

2023-11-25

Describe your issue very briefly here. Then show it with a minimal, self-contained example in the following R chunk.

```
knitr::opts_chunk$set(echo = TRUE)
# if you are using libraries, it's good practice to load them here
library(data.table)
```

```
## Warning: 程辑包'data.table'是用R版本4.3.1 来建造的
```

```
library(quadprog)
```

```
## Warning: 程辑包'quadprog'是用R版本4.3.1 来建造的
```

```
load_digits <- function(subset=NULL, normalize=TRUE) {

#Load digits and labels from digits.csv.

#Args:
#subset: A subset of digit from 0 to 9 to return.
#If not specified, all digits will be returned.
#normalize: Whether to normalize data values to between 0 and 1.

#Returns:
#digits: Digits data matrix of the subset specified.
#The shape is (n, p), where
#n is the number of examples,
#p is the dimension of features.
#labels: Labels of the digits in an (n, ) array.
#Each of label[i] is the label for data[i, :]

# load digits.csv, adopted from sklearn.

df <- fread("digits.csv")
df <- as.matrix(df)

## only keep the numbers we want.
if (length(subset)>0) {

  c <- dim(df)[2]
  l_col <- df[,c]
  index = NULL

  for (i in 1:length(subset)){

    number = subset[i]
    index = c(index,which(l_col == number))
  }
  sort(index)
  df = df[index,]
}

# convert to arrays.
digits = df[,-1]
labels = df[,c]

# Normalize digit values to 0 and 1.
if (normalize == TRUE) {
  digits = digits - min(digits)
digits = digits/max(digits)}


# Change the labels to 0 and 1.
for (i in 1:length(subset)) {
  labels[labels == subset[i]] = i-1
```

```
}

return(list(digits, labels))

}
```

```
split_samples <- function(digits,labels) {

# Split the data into a training set (70%) and a testing set (30%).

num_samples <- dim(digits)[1]
num_training <- round(num_samples*0.7)
indices = sample(1:num_samples, size = num_samples)
training_idx <- indices[1:num_training]
testing_idx <- indices[-(1:num_training)]

return (list(digits[training_idx,], labels[training_idx],
        digits[testing_idx,], labels[testing_idx]))
}
```

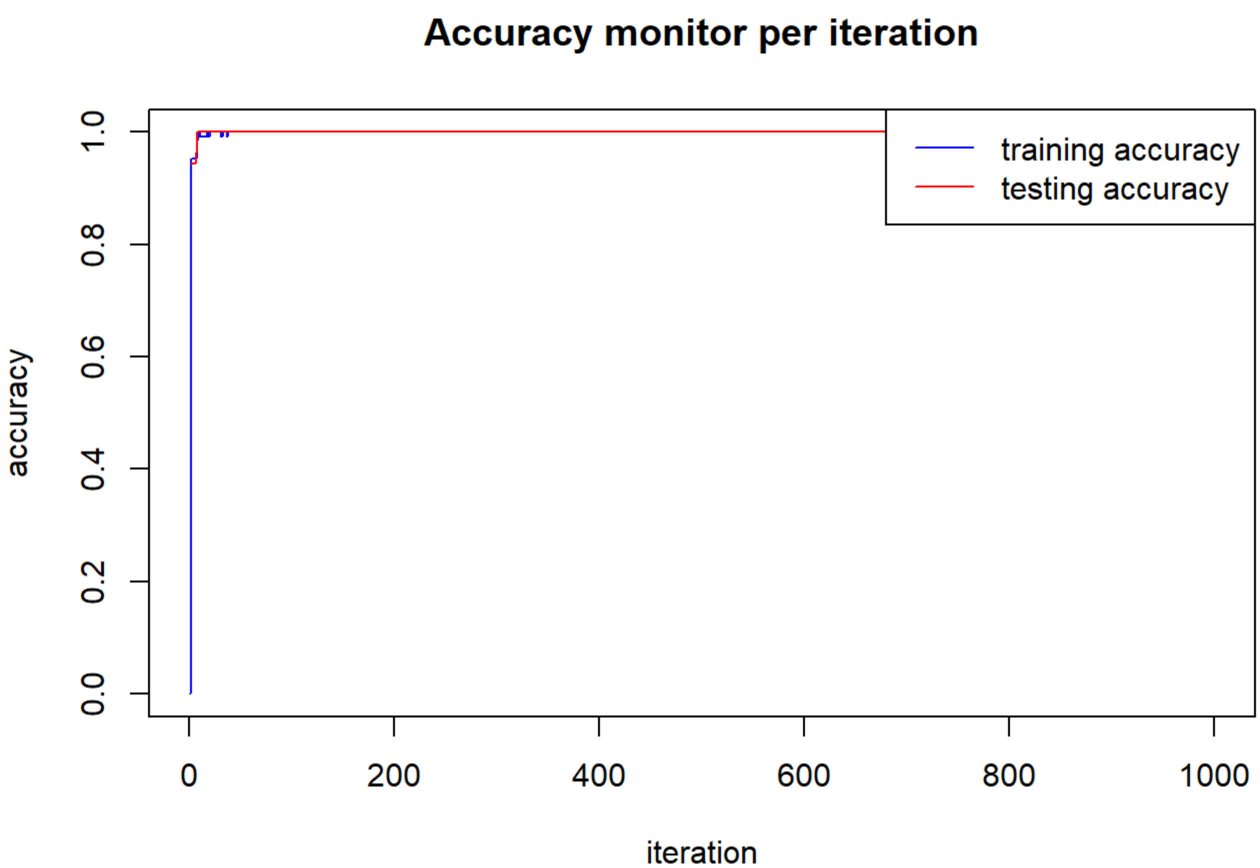
```
# Load digits and labels.
result = load_digits(subset=c(1, 7), normalize=TRUE)
digits = result[[1]]
labels = result[[2]]

result = split_samples(digits,labels)
training_digits = result[[1]]
training_labels = result[[2]]
testing_digits = result[[3]]
testing_labels = result[[4]]

# print dimensions
length(training_digits)
```

```
## [1] 16192
```

```
length(testing_digits)
```

```
## [1] 6912
```

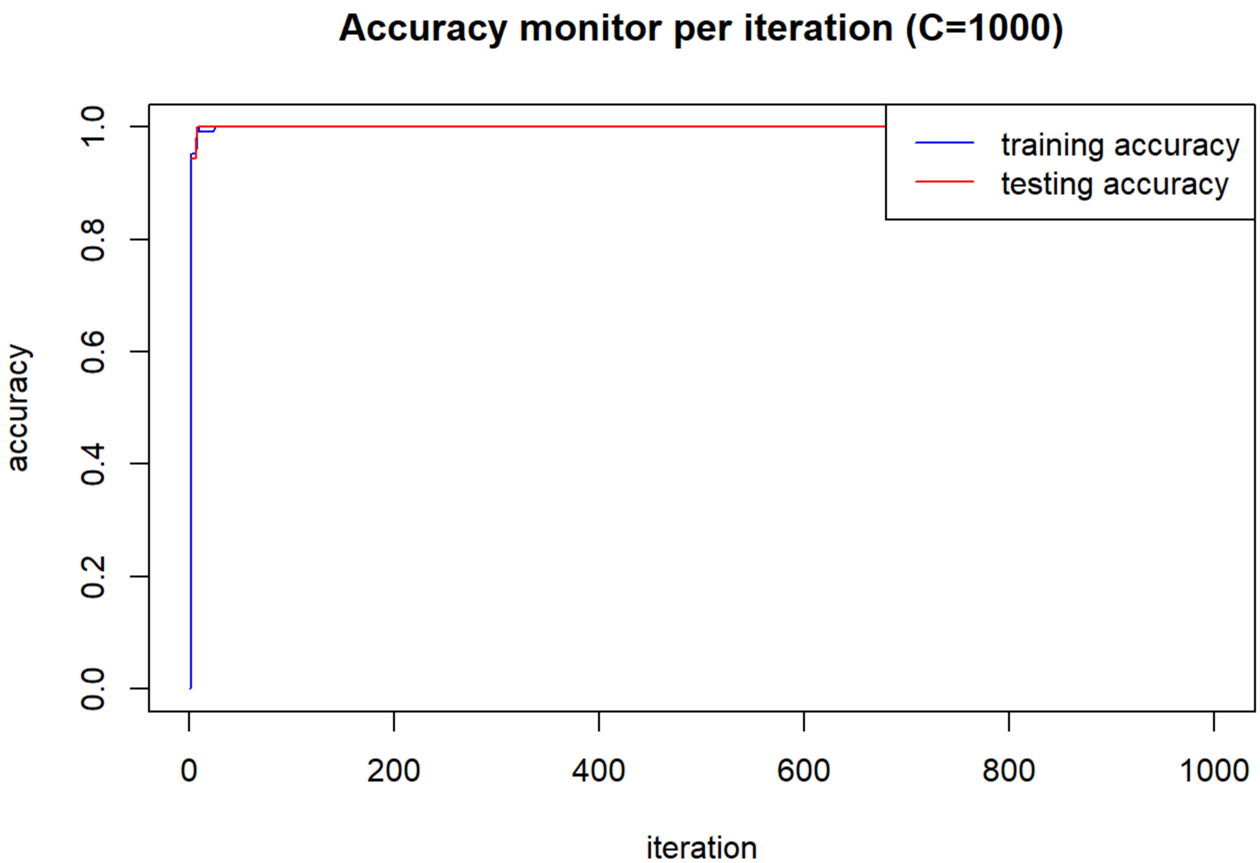
```
my_SVM <- function(X_train, Y_train, X_test, Y_test, lambda = 0.01,
                   num_iterations = 1000, learning_rate = 0.1)
{
  n <- dim(X_train)[1]
  p <- dim(X_train)[2] + 1
  X_train1 <- cbind(rep(1, n), X_train)
  Y_train <- 2 * Y_train - 1
  beta <- matrix(rep(0, p), nrow = p)
  ntest <- nrow(X_test)
  X_test1 <- cbind(rep(1, ntest), X_test)
  Y_test <- 2 * Y_test - 1
  acc_train <- rep(0, num_iterations)
  acc_test <- rep(0, num_iterations)
  for(it in 1:num_iterations)
  {
    s <- X_train1 %*% beta
    db <- s * Y_train < 1
    dbeta <- matrix(rep(1, n), nrow = 1) %*%((matrix(db*Y_train, n, p)*X_train1))/n;
    beta <- beta + learning_rate * t(dbeta)
    beta[2:p] <- beta[2:p] - lambda * beta[2:p]
    acc_train[it] <- mean(sign(s * Y_train))
    acc_test[it] <- mean(sign(X_test1 %*% beta * Y_test))
  }
  model <- list(beta = beta, acc_train = acc_train, acc_test = acc_test)
  model
}
model1 = my_SVM(training_digits,training_labels,testing_digits,testing_labels)
x = 1:1000
training_accuracy <- unlist(model1[2])
testing_accuracy <- unlist(model1[3])
plot(x, training_accuracy, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy", m
ain = "Accuracy monitor per iteration")
lines(x, testing_accuracy, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```



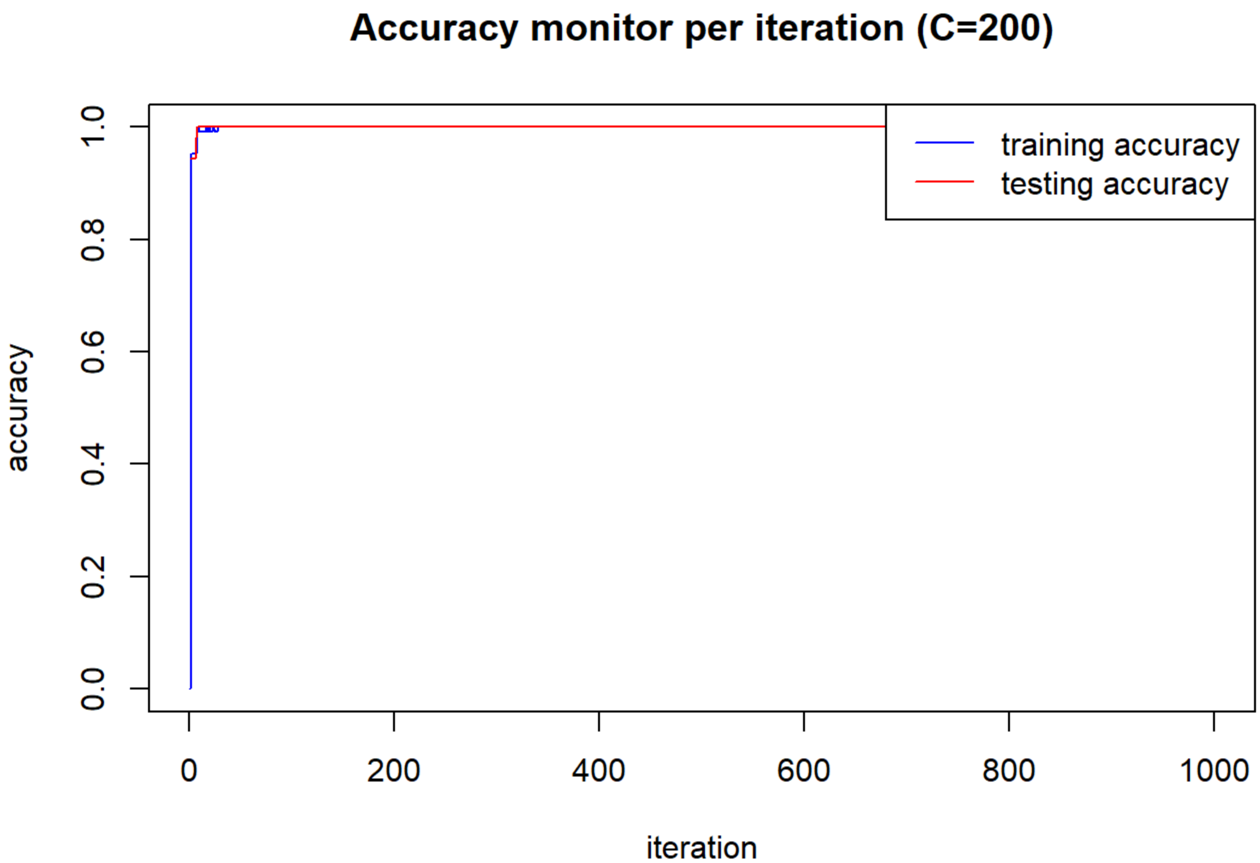
Problem 1

As shown above, the accuracy converges to 100% just after a few iterations (21 iterations for training accuracy and 7 iterations for testing accuracy). This suggests that SVM model performs quite well in this classification problem. Besides, Training accuracy starts from 0 in the first interation, while testing accuracy starts from a relatively high level.

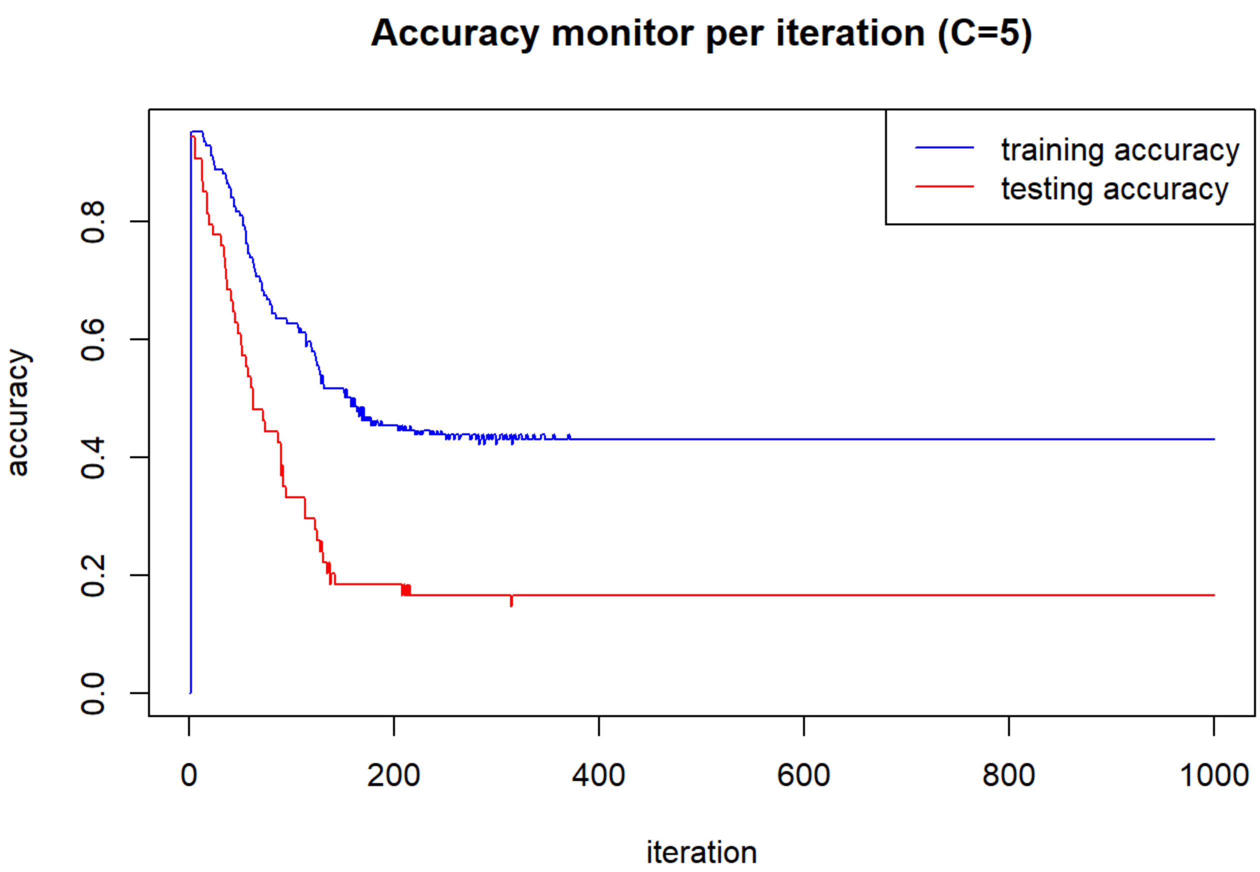
```
model2 = my_SVM(training_digits,training_labels,testing_digits,testing_labels,lambda=0.001)
training_accuracy2 <- unlist(model2[2])
testing_accuracy2 <- unlist(model2[3])
plot(x, training_accuracy2, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy",
main = "Accuracy monitor per iteration (C=1000)")
lines(x, testing_accuracy2, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```



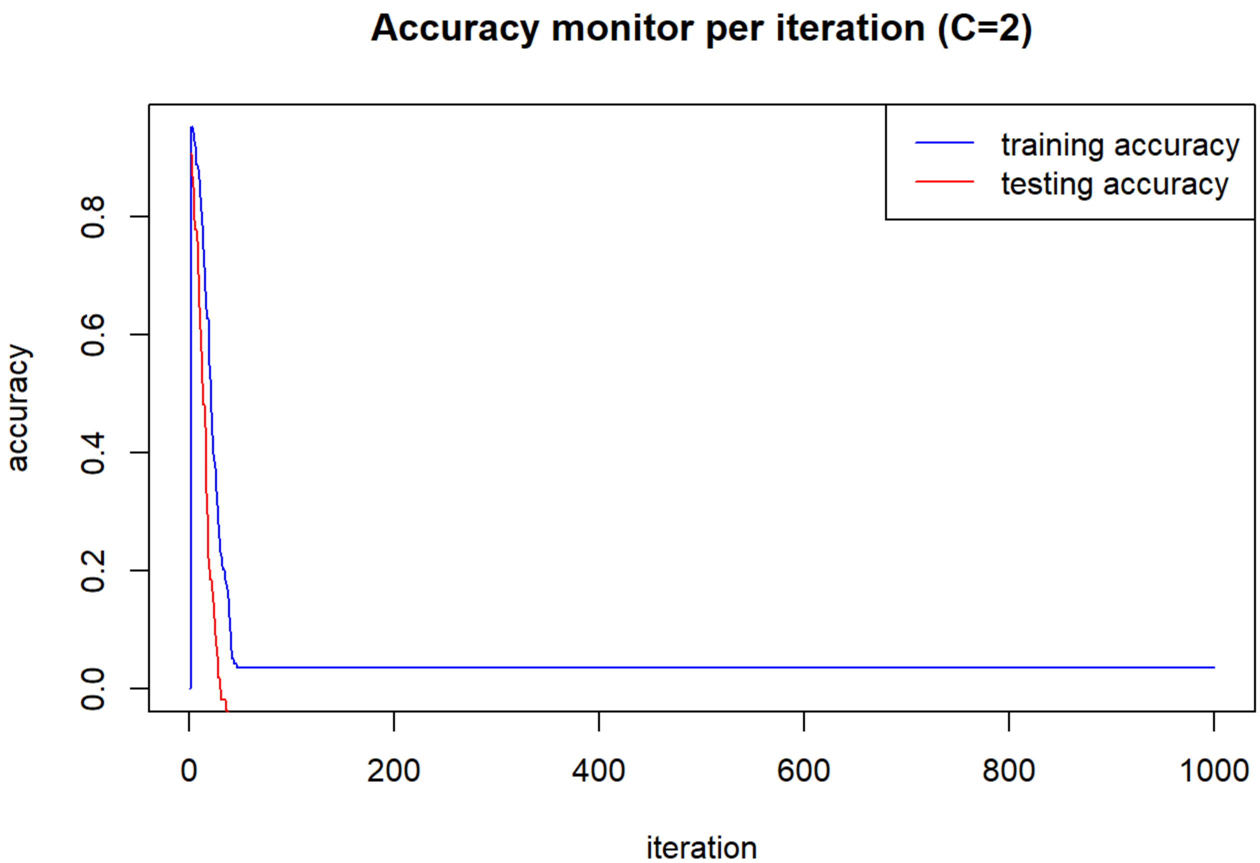

```
model3 = my_SVM(training_digits,training_labels,testing_digits,testing_labels,lambda=0.005)
training_accuracy3 <- unlist(model3[2])
testing_accuracy3 <- unlist(model3[3])
plot(x, training_accuracy3, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy",
main = "Accuracy monitor per iteration (C=200)")
lines(x, testing_accuracy3, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```



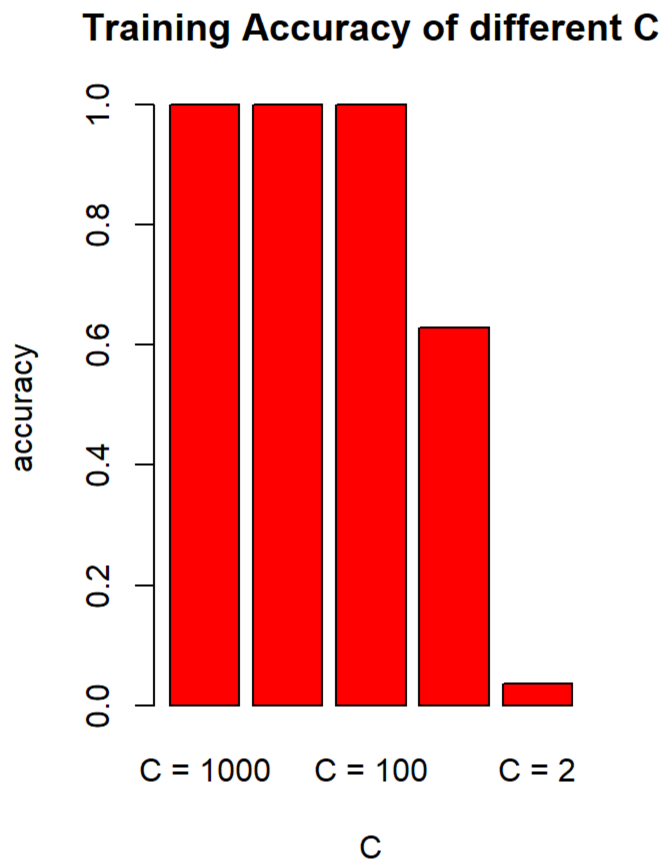

```
model4 = my_SVM(training_digits,training_labels,testing_digits,testing_labels,lambda=0.2)
training_accuracy4 <- unlist(model4[2])
testing_accuracy4 <- unlist(model4[3])
plot(x, training_accuracy4, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy",
main = "Accuracy monitor per iteration (C=5)")
lines(x, testing_accuracy4, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```

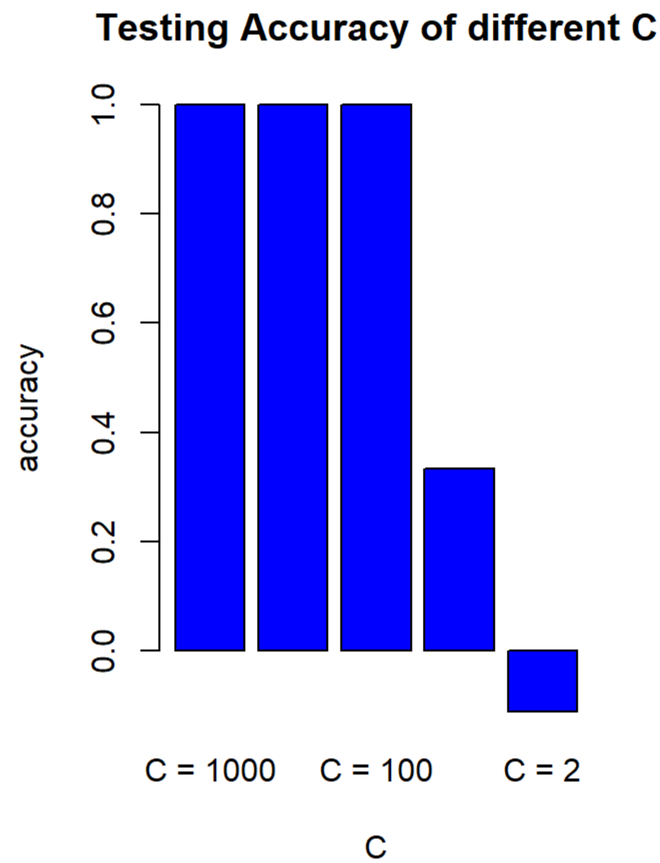



```
model5 = my_SVM(training_digits,training_labels,testing_digits,testing_labels,lambda=0.5)
training_accuracy5 <- unlist(model5[2])
testing_accuracy5 <- unlist(model5[3])
plot(x, training_accuracy5, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy",
main = "Accuracy monitor per iteration (C=2)")
lines(x, testing_accuracy5, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```



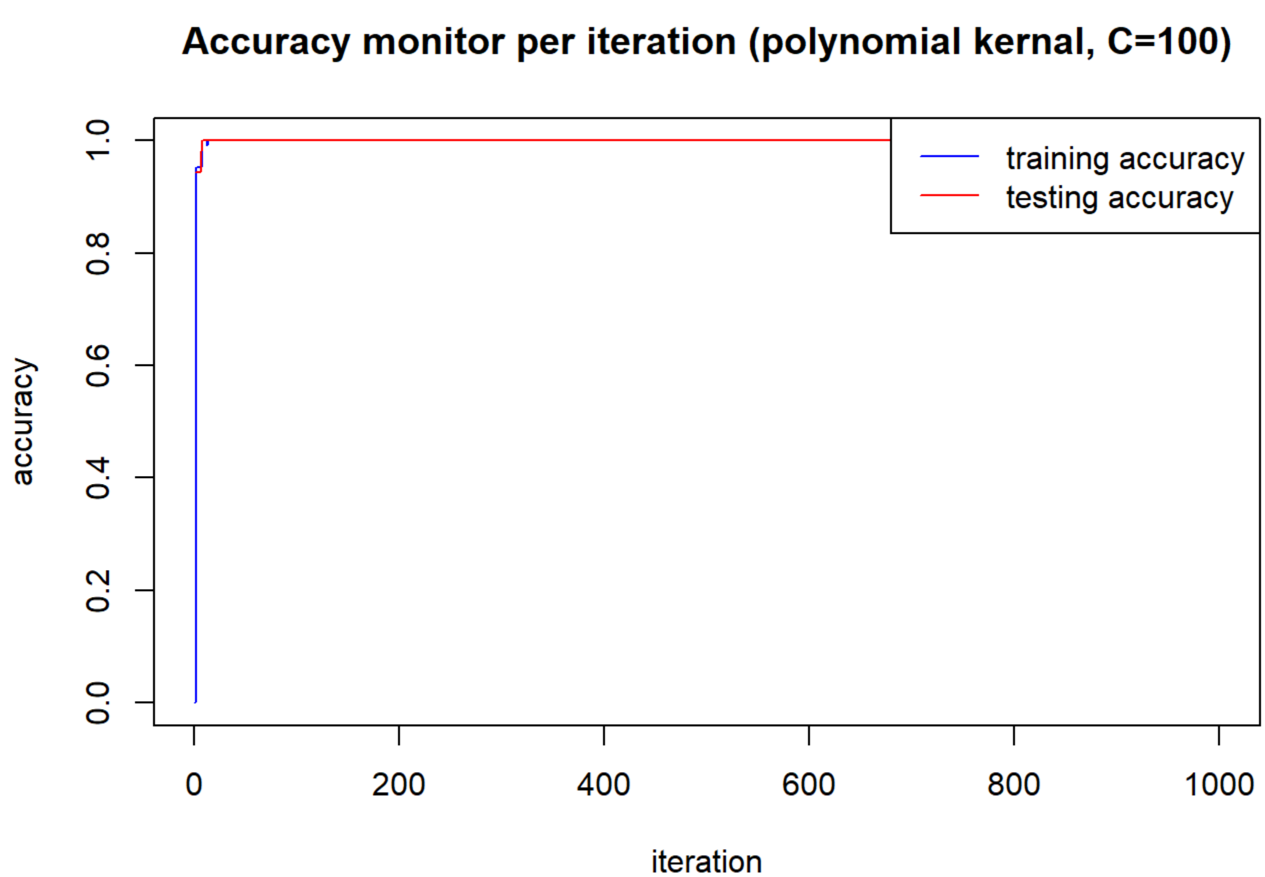

```
x <- c("C = 1000", "C = 200", "C = 100", "C = 5", "C = 2")
heights1 <- c(training_accuracy2[100],training_accuracy3[100],training_accuracy[100],training
_accuracy4[100],training_accuracy5[100])
heights2 <- c(testing_accuracy2[100],testing_accuracy3[100],testing_accuracy[100],testing_acc
uracy4[100],testing_accuracy5[100])

par(mfrow = c(1, 2))
barplot(heights1, names.arg = x, main = "Training Accuracy of different C", xlab = "C", ylab
= "accuracy", col = "red")
barplot(heights2, names.arg = x, main = "Testing Accuracy of different C", xlab = "C", ylab =
"accuracy", col = "blue")
```





Problem 2.a I selected lambda = 0.001, 0.005, 0.2 and 0.5 for comparisons.(Because the changing the slack variable C is equivalant to changing lambda). From the figures above, I found that small lambda will cause little impacts (just some random noises for training accuracy). However, if lambda gets larger(>0.1), there will be a steep decrease both for training accuracy and testing accuracy. I choose 100 iteration as my thershold, as typically the accuracy converges when the # of iterations reach 100. The result turns out that for lambda = 0.001, 0.005 and 0.01, the result is quite satisfying. While for lambda = 0.2 and lambda = 0.5, the accuracy decreases fast with the increase of lambda (or decrease of C).

```
polynomial <- function (X, degree=2){
  original <- as.matrix(X)
  n <- ncol(original)
  features <- list(original)
  for(i in range(n)){
    for(j in range(n)){
      features[[length(features)+1]] <- original[,i]*original[,j]
    }
  }
  features2 <- do.call(cbind,features)
  return(features2)
}
x <- 1:1000
training_digits2 <- polynomial(training_digits)
testing_digits2 <- polynomial(testing_digits)
model6 = my_SVM(training_digits2,training_labels,testing_digits2,testing_labels,lambda=0.01)
training_accuracy <- unlist(model6[2])
testing_accuracy <- unlist(model6[3])
plot(x, training_accuracy, type = "l", col = "blue", xlab = "iteration", ylab = "accuracy", m
ain = "Accuracy monitor per iteration (polynomial kernal, C=100)")
lines(x, testing_accuracy, col = "red")
legend("topright", legend = c("training accuracy", "testing accuracy"), col = c("blue", "re
d"), lty = 1)
```



Problem 2.b

As shown above, this is the training accuracy and testing accuracy with original features kerneled by second order polynomial, which is quite similar to the that of linear kernel, that is, testing accuracy and training accuracy converges to 100% after few periods of iterations.