STAT4060J hw4

Siqi Xiao

2022-11-01

Question 1

1. Write the R code linear SVM model to classify the digits. I attached the data file and a file of code to be completed. Please moniter the training accuracy and testing accuracy and describe your finding.

```
library("data.table")
load_digits <- function(subset=NULL, normalize=TRUE) {
#Load digits and labels from digits.csv.
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]))}
```

```r
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
dim(training_digits)
```

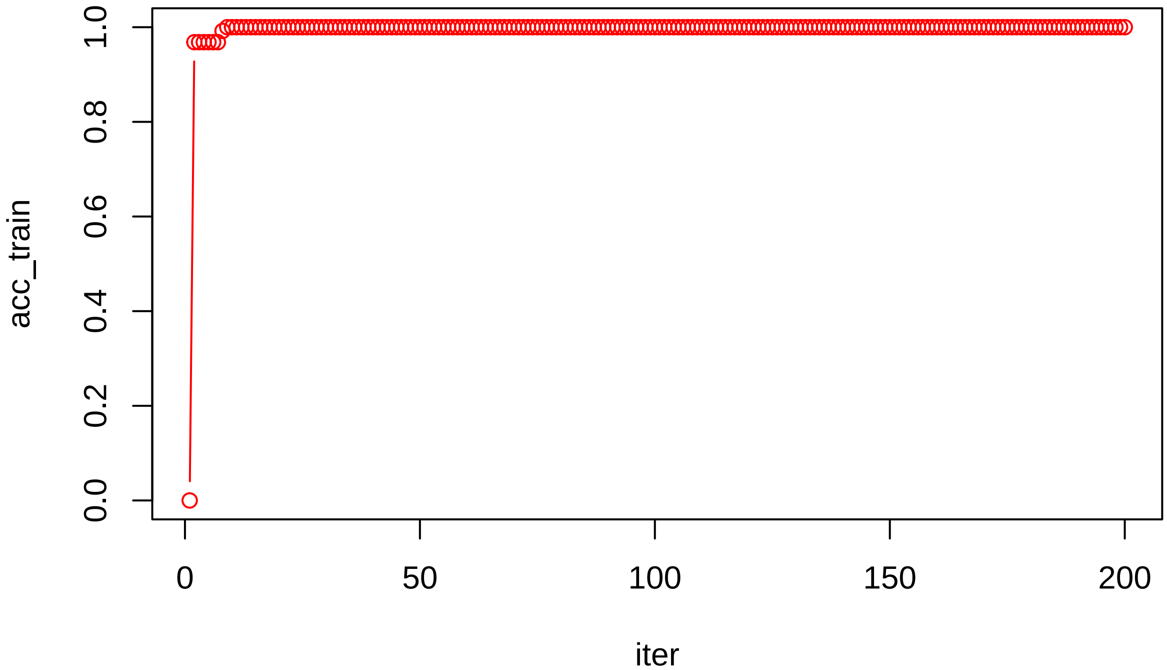
```
## [1] 253  64
```

```r
dim(testing_digits)
```

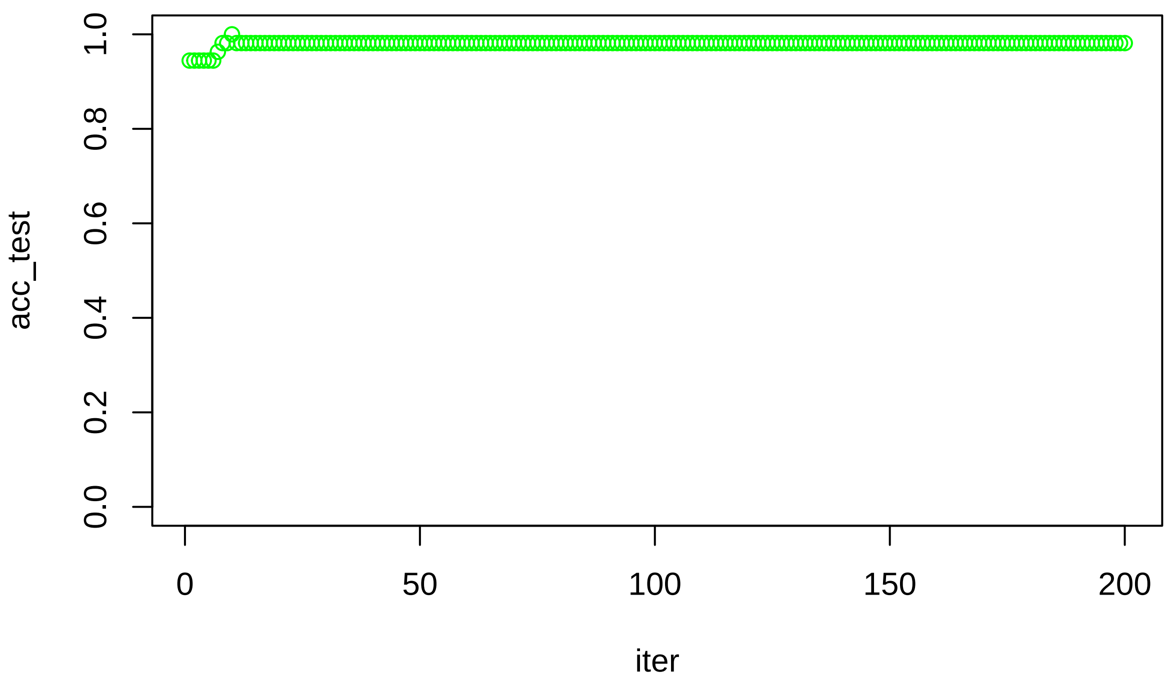
```
## [1] 108  64
```

```r
# Train a model and display training accuracy.
my_SVM <- function(X_train, Y_train, X_test, Y_test, lambda = 0.01, num_iterations = 1000, learning_rate
{
  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#0,1->-1,1
  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#0,1->-1,1

  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
}
```

```r
SVM_model <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 
beta <- SVM_model[[1]]
acc_train <- SVM_model[[2]]
acc_test <-  SVM_model[[3]]

iter <- 1:200
plot(iter, acc_train, type = 'b',ylim = c(0,1),col = 'red')
```




```
plot(iter, acc_test, type = 'b',ylim = c(0,1), col = 'green')
```



From the result above, I find that the accuracy for both training and testing are climbing in a fast speed, and finally come to a perfect accuracy. The first iteration is the iteration the model improves the fastest.

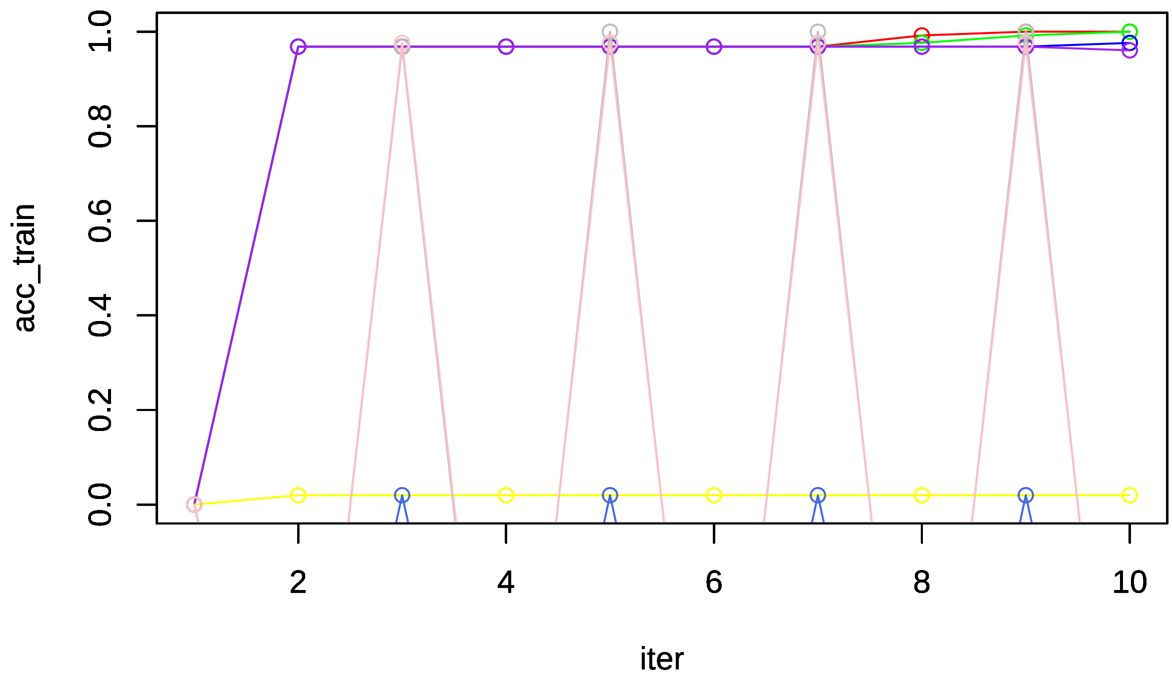
Question 2

2. Next, let's dive deeper into SVM, you should explore at least the following:

a. difference choice of C in controlling slackness
b. Kernel SVM with Gaussian radial basis function.
c. Another variation proposed by yourself.

```
library(plotrix)
model1 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model2 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model3 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model4 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model5 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model6 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model7 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model8 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model9 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10,
model10 <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterations = 10
```

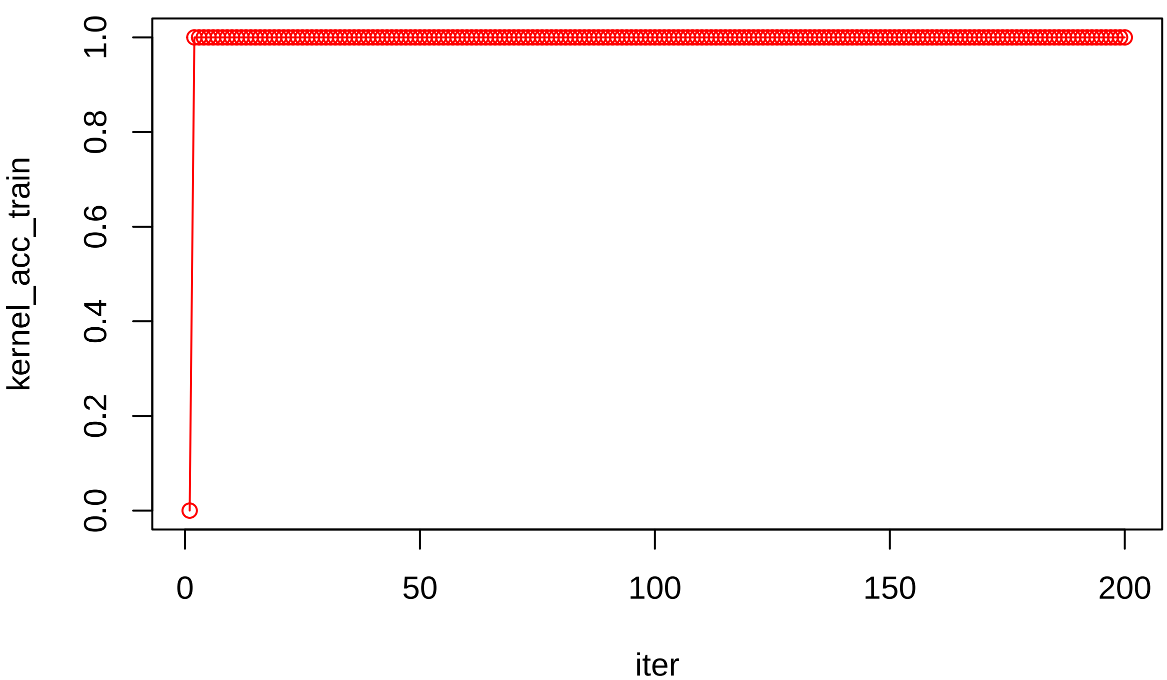
```
iter <- 1:10
acc_train1 <- model1[[2]]
acc_test1 <- model1[[3]]
acc_train2 <- model2[[2]]
acc_test2 <- model2[[3]]
acc_train3 <- model3[[2]]
acc_test3 <- model3[[3]]
acc_train4 <- model4[[2]]
acc_test4 <- model4[[3]]
acc_train5 <- model5[[2]]
acc_test5 <- model5[[3]]
acc_train6 <- model6[[2]]
acc_test6 <- model6[[3]]
acc_train7 <- model7[[2]]
acc_test7 <- model7[[3]]
acc_train8 <- model8[[2]]
acc_test8 <- model8[[3]]
acc_train9 <- model9[[2]]
acc_test9 <- model9[[3]]
acc_train10 <- model10[[2]]
acc_test10 <- model10[[3]]
plot(iter,acc_train1,type = 'o',ylim = c(0,1),col = "red",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train2,type = 'o',ylim = c(0,1),col = "green",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train3,type = 'o',ylim = c(0,1),col = "blue",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train4,type = 'o',ylim = c(0,1),col = "purple",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train5,type = 'o',ylim = c(0,1),col = "yellow",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train6,type = 'o',ylim = c(0,1),col = "orange",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train7,type = 'o',ylim = c(0,1),col = "springgreen",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train8,type = 'o',ylim = c(0,1),col = "royalblue",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train9,type = 'o',ylim = c(0,1),col = "grey",ylab = 'acc_train')
par(new = TRUE)
plot(iter,acc_train10,type = 'o',ylim = c(0,1),col = "pink",ylab = 'acc_train')
```



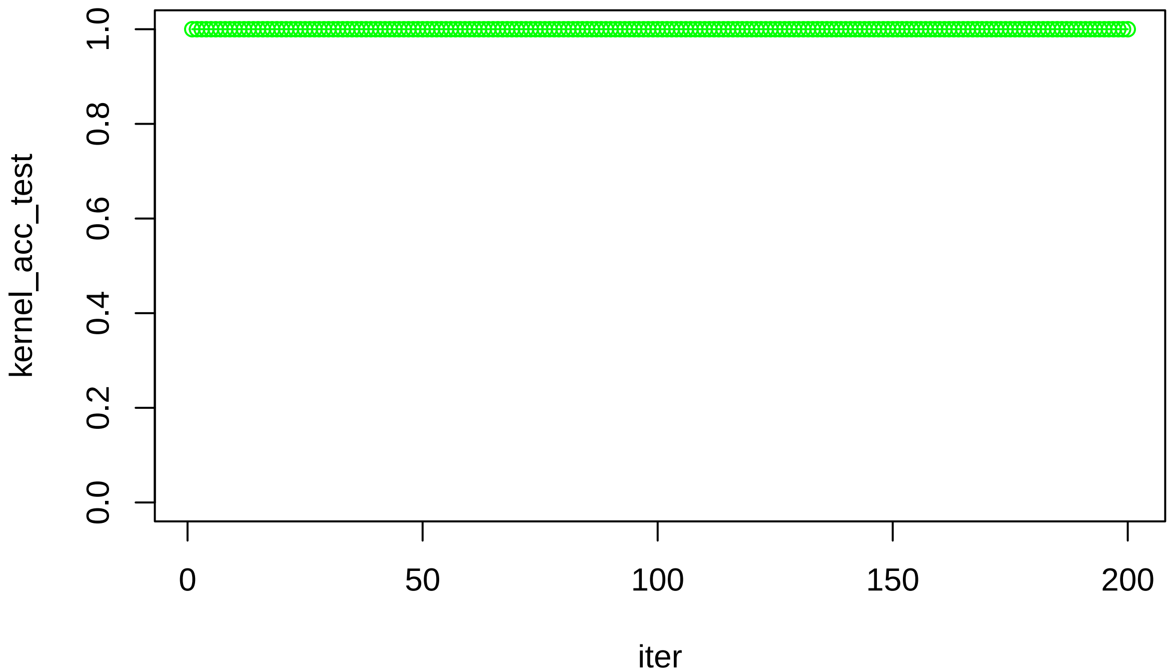
By choosing different Cs, the training accuracies and the testing accuracies have huge discrepancies. Generally speaking, from the experiment result, we may say the smaller the C is, the better the performance will be. With a large C, the SVM is not even able to have a fair performance on classification or regression.

```
rbf <- function(x1,x2,gamma){
  snorm <- sum((x1-x2)^2)
  result <- exp(-gamma*snorm)
  result
}
# use kernel trick
kernel_SVM <- function(X_train, Y_train, X_test, Y_test, lambda = 0.01, gamma = 0.5, num_iterations = 1(
{
  n <- dim(X_train)[1]
  #p <- dim(X_train)[2] + 1
  #X_train1 <- cbind(rep(1,n), X_train)
  K_matrix_train <- matrix(0, n, n)
  for (i in 1:n){
    for (j in 1:n){
      iden <- rbf(X_train[i,],X_train[j,],gamma)
      K_matrix_train[i,j] <- iden
    }
  }

  Y_train <- 2*Y_train - 1#0,1->-1,1
  beta <- matrix(rep(0,n), nrow = n)

  ntest <- nrow(X_test)
  #X_test1 <- cbind(rep(1,ntest), X_test)
  Y_test <- 2*Y_test - 1#0,1->-1,1
  K_matrix_test <- matrix(0, ntest, n)
  for (i in 1:ntest){
    for (j in 1:n){
      iden_test <- rbf(X_test[i,],X_train[j,],gamma)
```

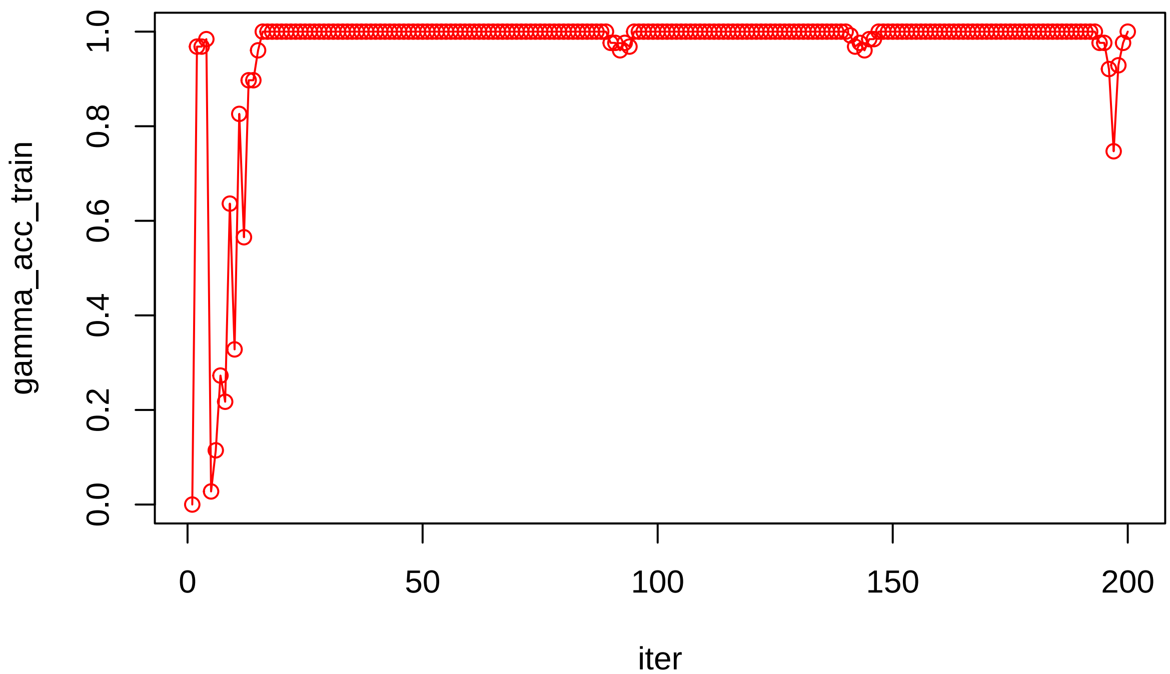
```r
      K_matrix_test[i,j] <- iden_test
    }
  }

  acc_train <- rep(0, num_iterations)
  acc_test <- rep(0, num_iterations)

  for(it in 1:num_iterations){
    s <- K_matrix_train %*% beta
    db <- s*Y_train < 1
    dbeta <- matrix(rep(1,n), nrow = 1)%*%((matrix(db*Y_train, n, n)*K_matrix_train))/n
    beta <- beta +learning_rate *t(dbeta)
    beta[1:n] <- beta[1:n] - lambda *beta[1:n]

    acc_train[it] <- mean(sign(s * Y_train))
    acc_test[it] <- mean(sign(K_matrix_test %*% beta * Y_test))
  }
  model <- list(beta = beta, acc_train = acc_train, acc_test = acc_test)
  model
}



kernel_model <- kernel_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iterat
kernel_beta <- kernel_model[[1]]
kernel_acc_train <- kernel_model[[2]]
kernel_acc_test <-  kernel_model[[3]]
iter <- 1:200
plot(iter, kernel_acc_train, type = 'o',ylim = c(0,1),col = 'red')
```



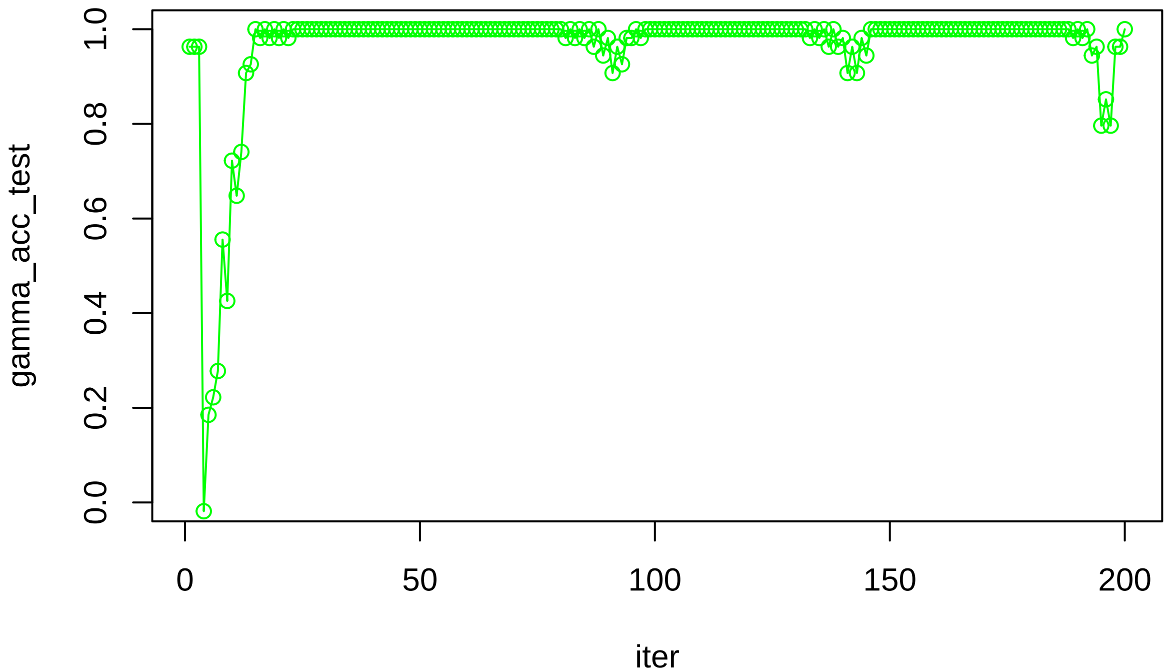
```
plot(iter, kernel_acc_test, type = 'o',ylim = c(0,1),col = 'green')
```




```
gamma_model <- kernel_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iteratio
gamma_beta <- gamma_model[[1]]
gamma_acc_train <- gamma_model[[2]]
gamma_acc_test <-  gamma_model[[3]]
iter <- 1:200

plot(iter , gamma_acc_train, type = 'o',ylim = c(0,1),col='red')
```

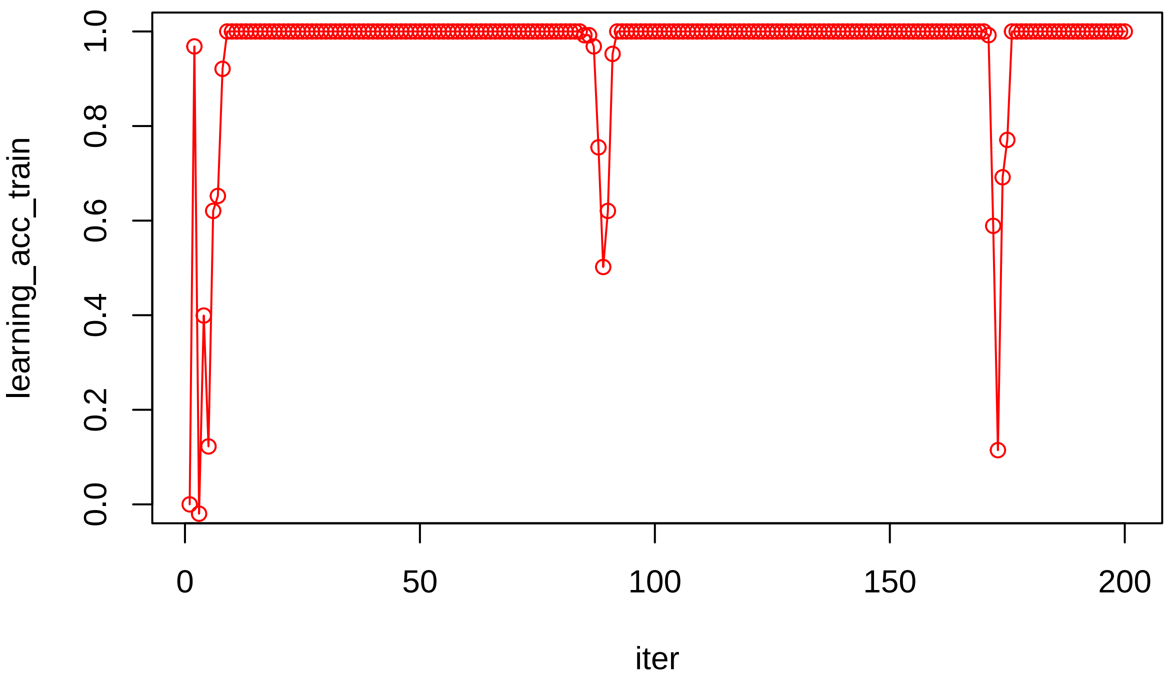


```
plot(iter , gamma_acc_test, type = 'o',ylim = c(0,1),col='green')
```

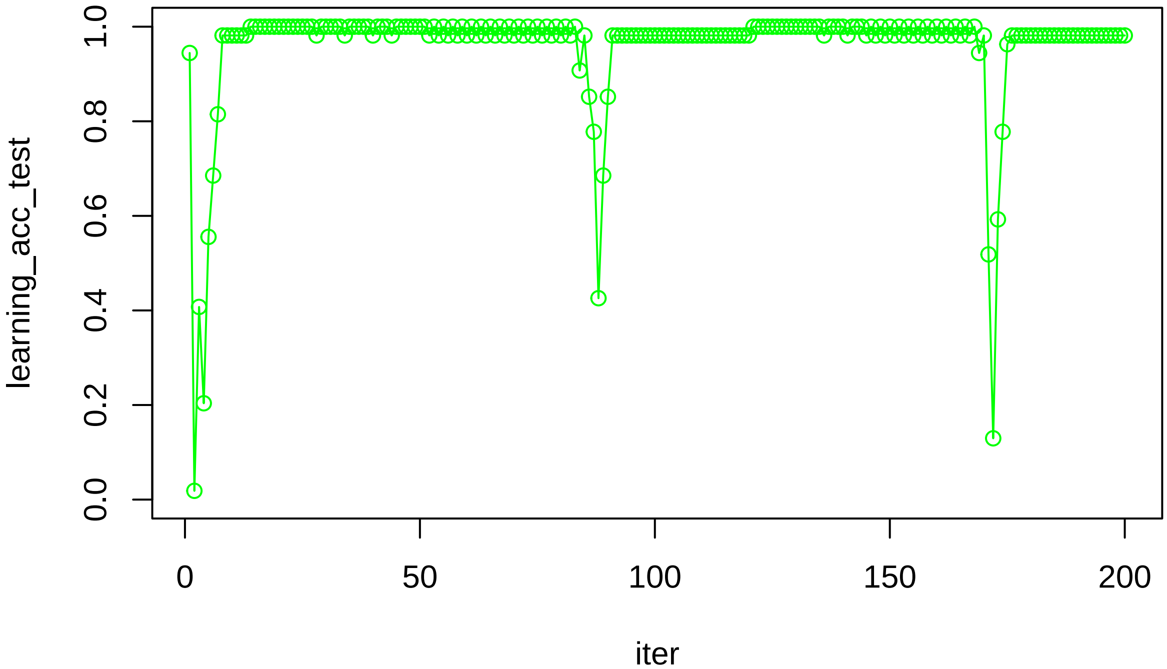


By changing different gamma of the kernel, we also get different accuracy. Larger gamma makes the model looser.

```
learning_model <- my_SVM(training_digits, training_labels, testing_digits, testing_labels, num_iteratio
learningl_beta <- learning_model[[1]]
learning_acc_train <- learning_model[[2]]
learning_acc_test <-  learning_model[[3]]
iter <- 1:200
plot(iter , learning_acc_train, type = 'o',ylim = c(0,1),col='red')
```



```r
plot(iter , learning_acc_test, type = 'o',ylim = c(0,1), col = 'green')
```



By changing learning rate to 1, we find the accuracy converges slower than lr=0.1, while the path is also not stable. But finally, the accuracy comes to a fair result. But such a large lr is not a great choice.