HW 5

Kang Yuan

2024-11-27

```
library(data.table) # allows us to use function fread,
# which quickly reads data from csv files


# load data
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
```

```
# Train a model and display training accuracy.
##### Put your work here
```

1.

```
hinge_loss <- function(w, X, y, C) {
  # Hinge loss in standard form
  # Args:
  # w: weight vector
  # X: input features (n x p matrix)
  # y: labels (-1 or 1)
  # C: penalty parameter for slack variables

  n <- nrow(X)
  margins <- 1 - y * (X %*% w)
  slack_loss <- sum(pmax(0, margins))
  regularization_loss <- 0.5 * sum(w^2)
  loss <- regularization_loss + C * slack_loss
  return(loss)
}

gradient <- function(w, X, y, C) {

  n <- nrow(X)
  margins <- 1 - y * (X %*% w)
  indicator <- ifelse(margins > 0, 1, 0)
  grad <- w - C * t(X) %*% (indicator * y) / n
  return(grad)
}

train_svm <- function(X, y, C=0.1, lr=0.1, epochs=1000) {
  # Train linear SVM using gradient descent
  # Args:
  # X: input features (n x p matrix)
  # y: labels (-1 or 1)
  # C: penalty parameter for slack variables
  # lr: learning rate
  # epochs: number of iterations

  n <- nrow(X)
  p <- ncol(X)
  w <- matrix(0, nrow=p, ncol=1)

  for (epoch in 1:epochs) {
    grad <- gradient(w, X, y, C)
    w <- w - lr * grad
    if (epoch %% 100 == 0) {
      cat("Epoch:", epoch, "Loss:", hinge_loss(w, X, y, C), "\n")
    }
  }
  return(w)
}

predict_svm <- function(X, w) {
  # Predict using the trained linear SVM
  # Args:
  # X: input features (n x p matrix)
  # w: weight vector

  preds <- X %*% w
  return(ifelse(preds >= 0, 1, -1))
}

training_labels_svm <- ifelse(training_labels == 1, 1, -1)
testing_labels_svm <- ifelse(testing_labels == 1, 1, -1)

training_digits <- cbind(1, training_digits)
testing_digits <- cbind(1, testing_digits)

C <- 0.1
lr <- 0.1
epochs <- 1000

weights <- train_svm(training_digits, training_labels_svm, C, lr, epochs)
```

```
## Epoch: 100 Loss: 22.75885
## Epoch: 200 Loss: 22.75878
## Epoch: 300 Loss: 22.75878
## Epoch: 400 Loss: 22.75878
## Epoch: 500 Loss: 22.75878
## Epoch: 600 Loss: 22.75878
## Epoch: 700 Loss: 22.75878
## Epoch: 800 Loss: 22.75878
## Epoch: 900 Loss: 22.75878
## Epoch: 1000 Loss: 22.75878
```

```
training_preds <- predict_svm(training_digits, weights)
training_accuracy <- mean(training_preds == training_labels_svm)
cat("Training Accuracy:", training_accuracy * 100, "%\n")
```

```
## Training Accuracy: 97.62846 %
```

```
testing_preds <- predict_svm(testing_digits, weights)
testing_accuracy <- mean(testing_preds == testing_labels_svm)
cat("Testing Accuracy:", testing_accuracy * 100, "%\n")
```

```
## Testing Accuracy: 98.14815 %
```

Findings: - The model achieves reasonable accuracy on both training and testing data. - The accuracy depends on the choice of C, learning rate and epochs. - More parameter tuning or feature engineering could improve results.

2.

a.

Effect of different C values on slackness control:

```
train_svm <- function(X, y, C=0.1, lr=0.1, epochs=1000) {

  n <- nrow(X)
  p <- ncol(X)
  w <- matrix(0, nrow=p, ncol=1)

  for (epoch in 1:epochs) {
    grad <- gradient(w, X, y, C)
    w <- w - lr * grad
  }
  return(w)
}

C_values <- c(0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50)

results <- data.frame(C = C_values, TrainingAccuracy = NA, TestingAccuracy = NA)

for (i in seq_along(C_values)) {
  C <- C_values[i]
  weights <- train_svm(training_digits, training_labels_svm, C, lr, epochs)

  training_preds <- predict_svm(training_digits, weights)
  training_accuracy <- mean(training_preds == training_labels_svm)

  testing_preds <- predict_svm(testing_digits, weights)
  testing_accuracy <- mean(testing_preds == testing_labels_svm)

  results$TrainingAccuracy[i] <- training_accuracy * 100
  results$TestingAccuracy[i] <- testing_accuracy * 100
}

print(results)
```

```
##        C TrainingAccuracy TestingAccuracy
## 1   0.01         97.62846        98.14815
## 2   0.10         97.62846        98.14815
## 3   0.20         97.62846        98.14815
## 4   0.50         98.41897        99.07407
## 5   1.00        100.00000       100.00000
## 6   2.00         92.49012        93.51852
## 7   5.00         88.14229        88.88889
## 8  10.00        100.00000       100.00000
## 9  20.00         84.18972        82.40741
## 10 50.00         70.35573        72.22222
```

b.

```
# Exponential (RBF) Kernel
exponential_kernel <- function(X, Z, gamma = 0.1) {
  n <- nrow(X)
  m <- nrow(Z)
  dist_matrix <- matrix(0, nrow = n, ncol = m)
  for (i in 1:n) {
    for (j in 1:m) {
      dist_matrix[i, j] <- sum((X[i, ] - Z[j, ])^2)
    }
  }
  return(exp(-gamma * dist_matrix))
}


hinge_loss_kernel <- function(alpha, K, y, C) {
  n <- length(y)
  dual_loss <- sum(alpha) - 0.5 * sum((y * alpha) %*% K %*% (y * alpha))
  regularization <- 0.5 * sum(alpha^2)
  return(-dual_loss + C * regularization)
}

gradient_kernel <- function(alpha, K, y, C) {
  grad <- rep(1, length(alpha)) - (y * alpha) %*% K * y
  return(grad)
}

train_kernel_svm <- function(X, y, C, kernel_function, gamma, lr = 0.1, epochs = 1000) {

  n <- nrow(X)
  alpha <- rep(0, n)
  K <- kernel_function(X, X, gamma)

  for (epoch in 1:epochs) {
    grad <- gradient_kernel(alpha, K, y, C)
    alpha <- alpha - lr * grad
    alpha <- pmax(0, alpha)
  }
  return(list(alpha = alpha, kernel_matrix = K))
}

predict_kernel_svm <- function(X_train, X_test, alpha, y_train, kernel_function, gamma) {

  K_test <- kernel_function(X_test, X_train, gamma)
  preds <- K_test %*% (alpha * y_train)
  return(ifelse(preds >= 0, 1, -1))
}

gamma <- 0.01
C <- 1.0
lr <- 0.1
epochs <- 1000

model <- train_kernel_svm(training_digits, training_labels_svm, C, exponential_kernel, gamma, lr, epochs)

training_preds <- predict_kernel_svm(training_digits, training_digits, model$alpha, training_labels_svm, exponent
ial_kernel, gamma)
training_accuracy <- mean(training_preds == training_labels_svm)
cat("Training Accuracy with Exponential Kernel:", training_accuracy * 100, "%\n")
```

```
## Training Accuracy with Exponential Kernel: 48.22134 %
```

```
testing_preds <- predict_kernel_svm(training_digits, testing_digits, model$alpha, training_labels_svm, exponentia
l_kernel, gamma)
testing_accuracy <- mean(testing_preds == testing_labels_svm)
cat("Testing Accuracy with Exponential Kernel:", testing_accuracy * 100, "%\n")
```

```
## Testing Accuracy with Exponential Kernel: 52.77778 %
```

Findings: - The model achieves poor accuracy on both training and testing data. - The accuracy seems not dependent on the choice of C, learning rate and epochs. - It seems that the original linear classification fits the training and prediction dataset.