Homework 2

CS 498, Spring 2018, Xiaoming Ji

The UC Irvine machine learning data repository hosts a collection of data on adult income, donated by Ronny Kohavi and Barry Becker. You can find this data at https://archive.ics.uci.edu/ml/datasets/Adult For each record, there is a set of continuous attributes, and a class "less than 50K" or "greater than 50K". There are 48842 examples. You should use only the continuous attributes (see the description on the web page) and drop examples where there are missing values of the continuous attributes. Separate the resulting dataset randomly into 10% validation, 10% test, and 80% training examples.

We will write a program to train a support vector machine on this data using stochastic gradient descent. We will ignore the id number, and use the continuous variables as a feature vector. You should scale these variables so that each has unit variance. We will search for an appropriate value of the regularization constant, trying at least the values [1e-3, 1e-2, 1e-1, 1]. Use the validation set for this search. We will use at least 50 epochs of at least 300 steps each. In each epoch, we will separate out 50 training examples at random for evaluation (call this the set held out for the epoch). We will compute the accuracy of the current classifier on the set held out for the epoch every 30 steps. We will produce:

Firstly, we load and process data per the instructions.

```
library(readr)
set.seed(19720816)

#Read data from 2 data files
data1 = read.csv("./data/adult.data", header = FALSE)[,c(1, 3, 5, 11, 12, 13, 15)]
data2 = read.csv("./data/adult.test", header = FALSE, skip = 1)[,c(1, 3, 5, 11, 12, 13, 15)]
```

Checking whether there are missing values.

```
sum(!complete.cases(data1))
```

```
## [1] 0
```

```
sum(!complete.cases(data2))
```

```
## [1] 0
```

- *No missing value found*

Merge these data and scale these variables so that each has unit variance. Then calculate the test/train split.

```
data1[, 7] = as.numeric(data1[, 7])
data1[data1[, 7] == 1, 7] = -1
data1[data1[, 7] == 2, 7] = 1

data2[, 7] = as.numeric(data2[, 7])
data2[data2[, 7] == 1, 7] = -1
data2[data2[, 7] == 2, 7] = 1

adult_data = rbind(data1, data2)

#Normalize the data
for (i in 1:6) {
  s = sd(adult_data[,i])
  m = mean(adult_data[,i])
```

```
  adult_data[,i] = (adult_data[,i] - m) / s
}

#Calculate train/validation/test split
total_count = dim(adult_data)[1]
test_count = round(total_count * 0.1)
valid_count = round(total_count * 0.1)

test_indexes = sample(1:total_count,test_count)
```

We will write a function to do Stochastic Gradient Descent, by using the following formula. Its extension to text book's example and add support to multiple sample N_b.

$$a^{(n+1)} = a^n - \frac{\eta}{N_b}(\sum_i^{N_b} \begin{cases} 0 & \text{if } y_i(a^T x_i + b) \geq 1 \\ -y_i x_i & otherwise \end{cases} + \lambda a)$$

$$b^{(n+1)} = b^n - \frac{\eta}{N_b} \sum_i^{N_b} \begin{cases} 0 & \text{if } y_i(a^T x_i + b) \geq 1 \\ -y_i & otherwise \end{cases}$$

Now, let's write the function.

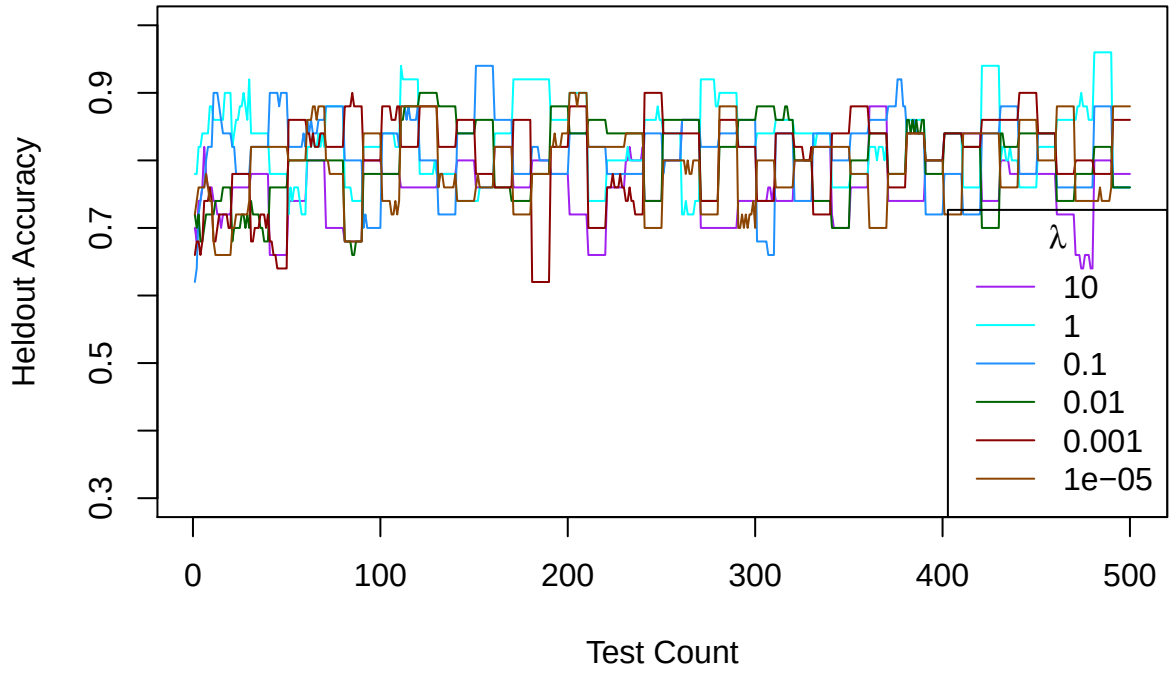
```
predict = function(a, b, features) {
  pred_y = c(a %*% t(features) + b)
  pred_y[pred_y <= 0] = -1
  pred_y[pred_y > 0] = 1

  return (pred_y)
}

sgd = function(epoch, n_steps, sample_count, valid_count, heldout_count, steps_before_testing, data, pa
  feature_count = dim(data)[2] - 1
  accuracy = c()
  w = c()
  v_a = parameters[1:feature_count]
  b = parameters[feature_count + 1]
  if(heldout_count > 0){
    heldout_accuracies = rep(0, epoch * n_steps / steps_before_testing)
    coef_magnitude     = rep(0, length(heldout_accuracies))
  } else {
    heldout_accuracies = NULL
    coef_magnitude     = NULL
  }

  if(valid_count > 0){
    valid_accuracies   = rep(0, epoch)
  } else {
    valid_accuracies   = NULL
  }

  total_step = 1
  test_count = 1

  for (epoch in 1:epoch) {
    #split heldout_data
```

```
    if(heldout_count > 0){
      heldout_indexes = sample(1: dim(data)[1], heldout_count)
      heldout_data = data[heldout_indexes,]
      #make the remaining the test and validation data
      r_indexes     = (1: dim(data)[1])[-heldout_indexes]
    } else {
      r_indexes     = (1: dim(data)[1])
    }

    train_indexes = sample(1:length(r_indexes), length(r_indexes) - valid_count)
    train_data  = data[r_indexes[train_indexes], ]
    if(valid_count > 0){
      valid_data  = data[r_indexes[-train_indexes],]
    }

    steplength  = 1 / (0.01 * epoch + 50)

    for (step in 1:n_steps){
      sample_indexes = sample(1:dim(train_data)[1], sample_count)
      x = train_data[sample_indexes, 1:feature_count]
      y = train_data[sample_indexes, feature_count + 1]

      h = y * (v_a %*% t(x) + b)

      #update a and b
      v_a = v_a - steplength/sample_count * (colSums(-y[h < 1] * x[h < 1,]) + lambda * v_a)
      b   = b - steplength/sample_count * sum(-y[h < 1])

      if(heldout_count > 0 && total_step %% steps_before_testing == 0){
        pred_y = predict(v_a, b, heldout_data[, 1:feature_count])
        heldout_accuracies[test_count] = sum(pred_y == heldout_data[, feature_count + 1]) / heldout_cou
        coef_magnitude[test_count] = sqrt(t(v_a) %*% v_a)
        test_count = test_count + 1
      }
      total_step = total_step + 1
    }
    if(valid_count > 0) {
      #calculate accuracy on validation data
      pred_y = predict(v_a, b, valid_data[, 1:feature_count])
      valid_accuracies[epoch] =  sum(pred_y == valid_data[, feature_count + 1]) / length(pred_y)
    }
  }
  return (list(a = v_a, b = b,
               heldout_accuracies = heldout_accuracies,
               coef_magnitude = coef_magnitude,
               valid_accuracies = valid_accuracies))
}
```

A plot of the accuracy every 30 steps, for each value of the regularization constant

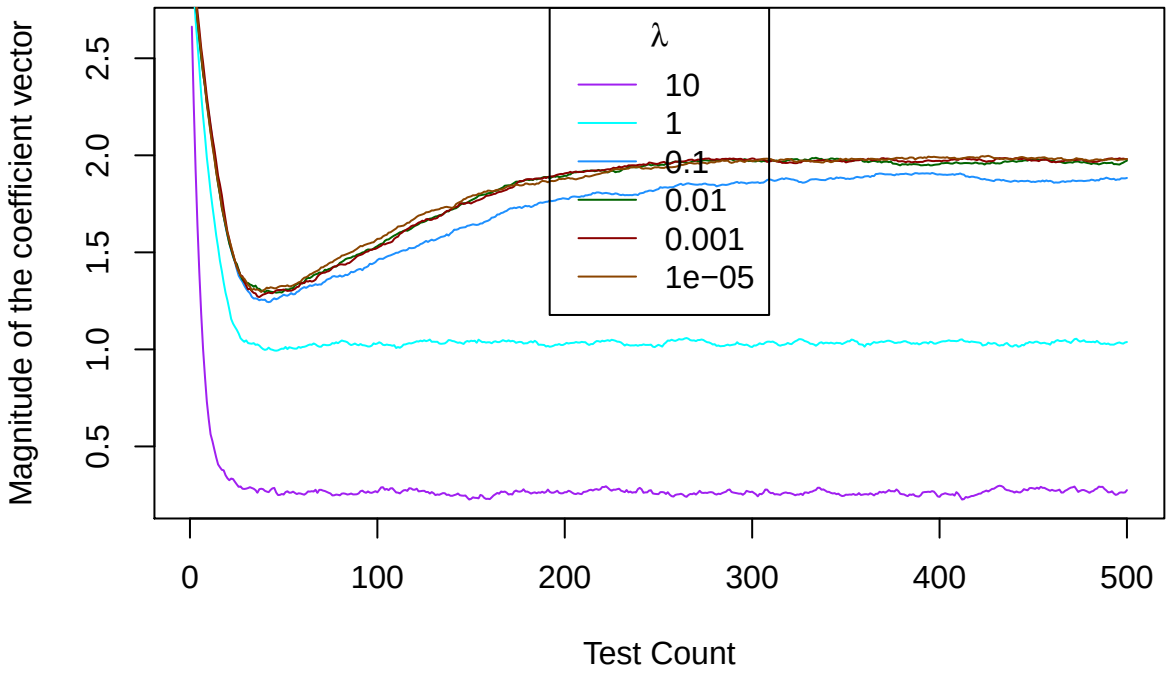
Note: we take 50 samples ($N_b = 30$) in order to make a good data coverage. Experiments also show lower the sample won't make much difference.

```
lambdas =  c(10, 1, 1e-1, 1e-2, 1e-3, 1e-5)
params = runif(7, 0, 2)
results = list()

for (i in 1:length(lambdas)){
  results[[i]] = sgd(50, 300, 50, valid_count, 50, 30, adult_data[-test_indexes,], params, lambdas[i])
}

colors = c("purple", "cyan", "dodgerblue", "darkgreen", "darkred", "darkorange4")

plot(1:length(results[[1]]$heldout_accuracies), results[[1]]$heldout_accuracies, type = "l", ylim=c(0.3

for (i in 2:length(lambdas)) {
  lines(1:length(results[[i]]$heldout_accuracies), results[[i]]$heldout_accuracies, col=colors[i])
}

legend("bottomright", title = expression(lambda), legend = lambdas, lwd = 1, col = colors)
```



A plot of the magnitude of the coefficient vector every 30 steps, for each value of the regularization constant

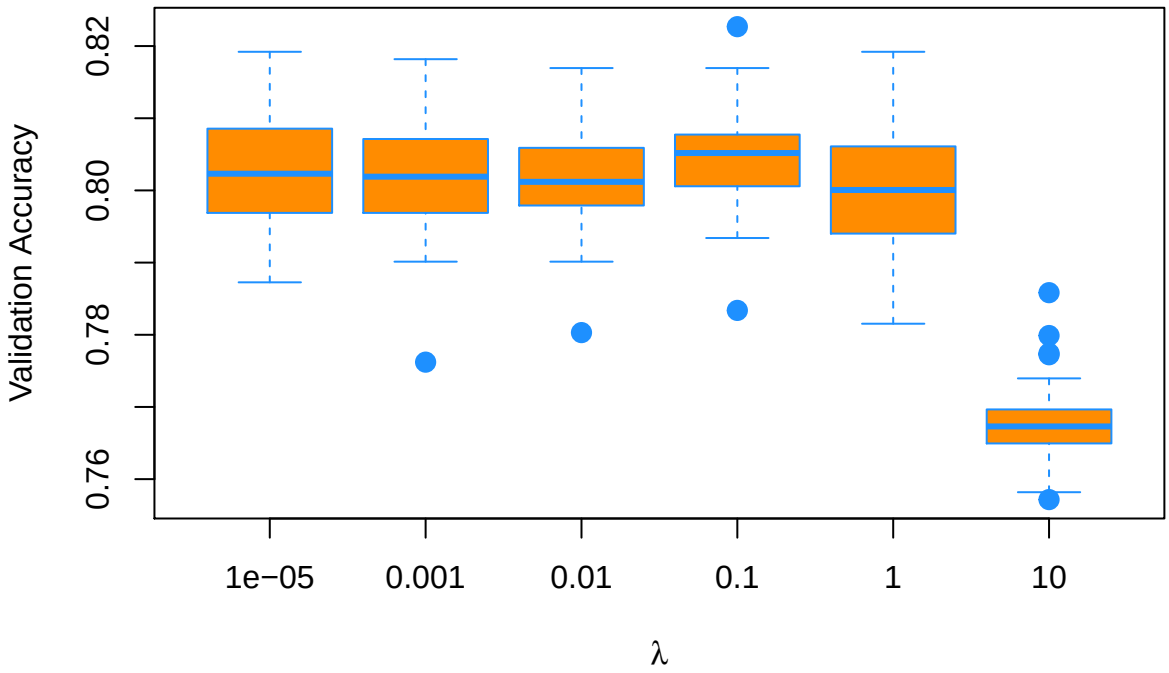
```
plot(1:length(results[[1]]$coef_magnitude), results[[1]]$coef_magnitude, type = "l", xlab="Test Count",

for (i in 2:length(lambdas)) {
  lines(1:length(results[[i]]$coef_magnitude), results[[i]]$coef_magnitude, col=colors[i])
}

legend("top", title = expression(lambda), legend = lambdas, lwd = 1, col = colors)
```



Estimate of the best value of the regularization constant (together with a brief description of why you believe that is a good value)

Let's make a boxplot of validation accuracy for each λ

```
df = data.frame()
for (i in 1:length(lambdas)){
  df = rbind(df, data.frame(y = results[[i]]$valid_accuracies, lambda=lambdas[i]))
}

boxplot(y ~ lambda, data = df, xlab=expression(lambda), ylab="Validation Accuracy", pch = 20, cex = 2, 
```



By checking this plot, $\lambda = 0.1$ has the best mean of validation accuracy and relatively smallest magnitude of the coefficient vector (small coefficient means small hinge loss for unseen data). Thus, we choose *0.1* as our best λ.

Estimate of the accuracy of the best classifier on the 10% test dataset data

```
params = runif(7, 0, 2)

model = sgd(50, 300, 50, 0, 0, 0, adult_data[-test_indexes,], params, 0.1)
pred_y = predict(model$a, model$b, adult_data[test_indexes, 1:6])
accuracy = sum(pred_y == adult_data[test_indexes, 7]) / length(pred_y)
```

Accuracy: 0.8040541