AML Assignment 2
Support Vector Machines

Team:

Julia Tcholakova (dt2@illinois.edu)

Apoorva Srinivasa (apoorva6@illinois.edu)

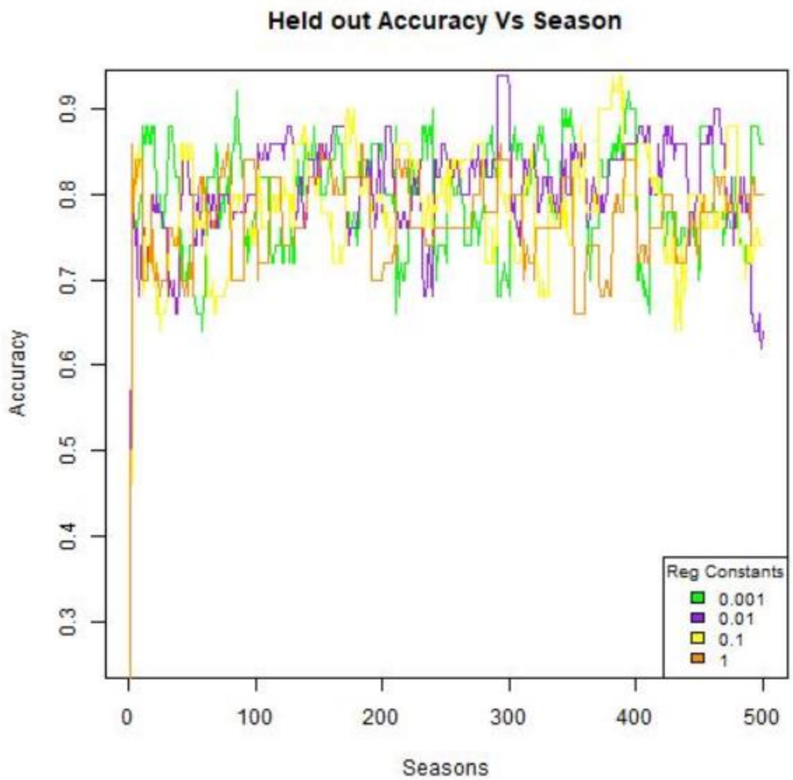
STUDENT
Apoorva Srinivasa

AUTOGRADER SCORE
81.7 / 100.0

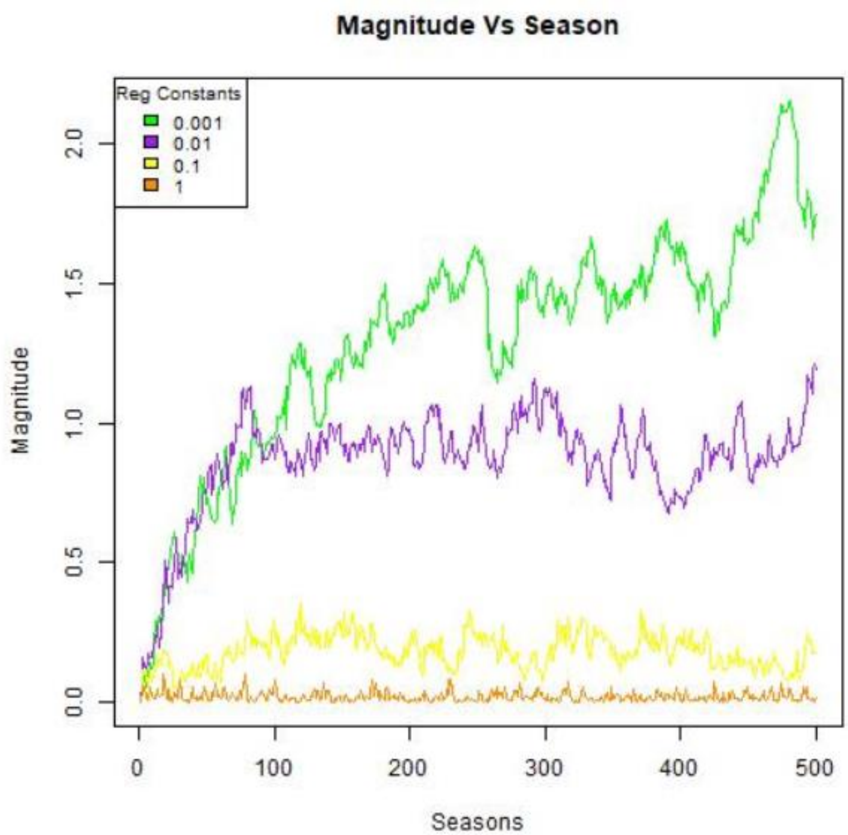
STUDENT
Iulia Tcholakova

AUTOGRADER SCORE
81.7 / 100.0

Plot of Accuracy Vs Seasons



Plot of Magnitude of a Vs Seasons



Estimate of the best value of the regularization constant

- Our estimate for the best lambda value of the regularization constant is 0.001.
- When comparing the accuracies generated with different lambdas on the validation set (0.001,0.01,0.1,1), the value of 0.001 generated the best accuracy.
- The accuracies of all four regularization constant values obtained during the experiment are shown below:

lambda	accuracy
0.001	0.8157416
0.01	0.8048226
0.1	0.7979982
1	0.7784349

As we can see, the value of **0.001** generated the best accuracy.

Choice of learning rate

- Our choice of learning rate was **0.0054**, this rate gave us the highest testing score
- Our approach was to try different values for the problem and see which works best
- We tried different options with small values between 0.01 and 0.009
- Here are some of the results for good performing rates:

learning rate	score
0.0054	81.7
0.0055	81.51
0.006	81.39
0.0014	80.27
0.0044	80.1

Code for SGD :

```
for (season in 1:seasons){    #please refer to p 6 for details
    ex_idx = sample(1:dim(train_x)[1], 50)
    ex_data = train_x[ex_idx, ]
    ex_labels = train_y[ex_idx]
    train_data = train_x[-ex_idx,]     #remove sample data
    train_labels = train_y[-ex_idx]
     #init num of steps
    num_steps = 0
    for (step in 1:steps){
      if(num_steps %% steps_eval == 0){
        calc = calcAccuracy(ex_data, ex_labels, a, b)
        acc_vec = c(acc_vec, calc[1])
        mag_vec = c(mag_vec, calcMagnitude(a))
      }
      k = sample(1:length(train_labels), 1)
      x_loc = as.numeric(as.matrix( train_data[k,] ))
      y_loc = train_labels[k]
       pred = calcGamma(x_loc, a, b)
      steplen = 1 / ((steplen_a * season) + steplen_b)
       #gradient
      if(y_loc * pred >= 1){
        p1 = lambdas[i] * a
        p2 = 0
        positive= positive+ 1
      } else {
        p1 = (lambdas[i] * a) - (y_loc * x_loc)
        p2 = -(y_loc)
        negative= negative+ 1
      }
      #next
      a = a - (steplen * p1)
      b = b - (steplen * p2)
      #accumulate to check at the end
      num_steps = num_steps + 1
    }
  }
```

Evaluating test data

```
for(i in 1:nrow(test_scaled_df)) {
  calculated_test = as.numeric((as.matrix(test_scaled_df[i,]) %*% t(a) ) + b )
  pred_test = ifelse(calculated_test >= 0, ">50K", "<=50K")
  test_label = c(test_label,pred_test)
}
```

All Code

```
#Homework 2
#Adult dataset SVM
library(caret)
#get the data
adult_train_all = read.csv("train.txt", header = FALSE)
adult_test_all = read.csv("test.txt", header = FALSE)
#extract only continous variables
train_df = adult_train_all[-c(2,4,6,7,8,9,10,14)]
#create the response variable
train_df$y = ifelse(train_df$V15 == " <=50K", -1, 1)
#has no y
test_df = adult_test_all[-c(2,4,6,7,8,9,10,14)]

#only leave the contious variables
#continuous variables: age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week
#train data
train_prep = train_df[-c(7,8)]
train_prep = scale(train_prep)
K_50 = train_df$V15
y = train_df$y
train_scaled_df = cbind(as.data.frame(train_prep), K_50, y)
#test data
test_prep = scale(test_df)
test_scaled_df = as.data.frame(test_prep)
#split train data, use 10% for validation set
#random split, get the index first
idx_train = sample(1:nrow(train_scaled_df), 0.9 * nrow(train_scaled_df))
idx_test = setdiff(1:nrow(train_scaled_df), idx_train)
#split the data to have a validation set as well
train_split_df = train_scaled_df[idx_train, ]
val_split_df = train_scaled_df[idx_test, ]
#prepare for processing
train_x = train_split_df[,-c(7,8)]
train_y = train_split_df[,8]
test_x = test_scaled_df
valid_x = val_split_df[, -c(7,8)]
valid_y = val_split_df[, 8]

#parameters as per assignment:
#suggested reg constant or lambdas [1e-3, 1e-2, 1e-1, 1] or [0.001, 0.01, 0.1, 1]
lambdas = c(1e-3, 1e-2, 1e-1, 1)
seasons = 50
```

```
steps = 300
#number of training examples at random for evaluation, call this the set held out for the season
num_eval_examples = 50
#need to compute the accuracy of the current classifier on the held out set for the season every
30 steps
steps_eval = 30
steplen_a = .01
steplen_b = 50

#calc gamma function
calcGamma = function(x, a, b){
  x_mat = as.numeric(as.matrix(x))
  return (t(a) %*% x_mat + b)
}
#assign label
assignLabel = function(x){
  if(x >= 0){
    return(1)
  }
  else{
    return(-1)
  }
}
#calculate accuracy
calcAccuracy = function(x,y,a,b){
  match = 0
  no_match = 0

  for (i in 1:length(y)){
    pred = calcGamma(x[i,], a, b)
    pred = assignLabel(pred)
    actual = y[i]

    if(pred == actual){
      match = match + 1
    } else{
      no_match = no_match + 1
    }
  }
  return(c( (match/(match+no_match)), match, no_match) )
}
#calculate magnitude
calcMagnitude = function(a){
  m = (t(a) %*% a)/2
```

```
  return(m)
}

#vectors to store
acc_valid_vec = rep(0, length(lambdas))
acc_test_vec = rep(0, length(lambdas))
#for the plots
acc_mat = matrix(nrow = 4, ncol = 500)
magnitude_mat = matrix(nrow = 4, ncol = 500)
magnitude_list = list()
#run for each lambda
for (i in 1:length(lambdas)){
  #init a and b
  a = c(0,0,0,0,0,0)
  b = 0
  acc_vec = c()
  mag_vec = c()
  positive = 0
  negative = 0

  for (season in 1:seasons){
    #50 random examples for evaluation after every 30 steps
    #get the rows for the sample
    ex_idx = sample(1:dim(train_x)[1], 50)
    ex_data = train_x[ex_idx, ]
    ex_labels = train_y[ex_idx]
    train_data = train_x[-ex_idx,]     #remove sample data
    train_labels = train_y[-ex_idx]

    #init num of steps
    num_steps = 0
    for (step in 1:steps){
      #check for remainder
      if(num_steps %% steps_eval == 0){
        calc = calcAccuracy(ex_data, ex_labels, a, b)
        acc_vec = c(acc_vec, calc[1])
        mag_vec = c(mag_vec, calcMagnitude(a))
      }

      k = sample(1:length(train_labels), 1)
      x_loc = as.numeric(as.matrix( train_data[k,] ))
      y_loc = train_labels[k]

      pred = calcGamma(x_loc, a, b)
```

```
      steplen = 1 / ((steplen_a * season) + steplen_b)

      #gradient
      if(y_loc * pred >= 1){
        p1 = lambdas[i] * a
        p2 = 0
        positive= positive+ 1
      } else {
        p1 = (lambdas[i] * a) - (y_loc * x_loc)
        p2 = -(y_loc)
        negative= negative+ 1
      }
      #next
      a = a - (steplen * p1)
      b = b - (steplen * p2)
      #accumulate to check at the end
      num_steps = num_steps + 1
    }
  }

  check = calcAccuracy (valid_x, valid_y, a, b)
  acc_valid_vec[i] = check[1]

  acc_mat[i,] = acc_vec
  magnitude_mat[i,] = mag_vec

}

##accuracy plot
accuracy_df = as.data.frame(acc_mat)
rownames(accuracy_df) = c("0.001","0.01", "0.1", "1")
accuracy_df = t(accuracy_df)
plot_num = nrow(accuracy_df)
jpeg(file=paste(toString("Accuracies  HW2"),".jpg") )
plot(1:plot_num, accuracy_df[, 1],  type ="l",col="green", main = "Held out Accuracy Vs
Season", xlab ="Seasons",ylab ="Accuracy")
lines(1:plot_num, accuracy_df[, 2], type="l", col="purple")
lines(1:plot_num, accuracy_df[, 3], type="l", col="yellow")
lines(1:plot_num, accuracy_df[, 4], type="l", col="darkorange")
legend("bottomright",c("0.001", "0.01", "0.1", "1"),
fill=c("green","purple","yellow","darkorange"),title="Reg Constants", cex = 0.8)
dev.off()

magnitude_df = as.data.frame(magnitude_mat)
```

```
rownames(magnitude_df) = c("0.001","0.01", "0.1", "1")
magnitude_df = t(magnitude_df)

##magnitude plot
jpeg(file=paste(toString("Magnitude  HW2"),".jpg") )
plot(1:plot_num, magnitude_df[, 1],  type ="l",col="green", main = "Magnitude Vs Season", xlab
="Seasons",ylab ="Magnitude")
lines(1:plot_num, magnitude_df[, 2], type="l", col="purple")
lines(1:plot_num, magnitude_df[, 3], type="l", col="yellow")
lines(1:plot_num, magnitude_df[, 4], type="l", col="darkorange")
legend("topleft",c("0.001", "0.01", "0.1", "1"),
fill=c("green","purple","yellow","darkorange"),title="Reg Constants", cex = 0.8)
dev.off()
#######################
####testing############
train_scaled = as.data.frame(train_prep)
train_scaled$y = train_df$y
best_lamda = 0.001
#Initialising vector a and b
a = matrix(runif(n=6, min=-1, max=0), nrow = 1, ncol = 6)
b = 0.25 #B 0.2 (81.88) 0.25(82.13)

for(season in  1:seasons)
{
  eta = 0.0054   ##0.008 (81.08) 0.006 (81.62) 0.005 (80.20)  0.001 (79.16) 0.007 (80.70) 0.0055
(81.17)  0.0062 (81.43) 0.0065 (80.68) 0.0061(81.11)
  for (step in 1:num_steps)
  {
    #  Number of examples in batch = 1
    sample1 = sample(1:nrow(train_scaled), 1, replace = FALSE)
    x = as.matrix(train_scaled[sample1,-7])
    y = train_scaled[sample1,7]

    pred = y * ((x %*% t(a) + b))

    if(pred >= 1){
      a = a - (eta * best_lamda * a)
      b = b
    }
    else{
      a = a - (eta * ((best_lamda * a) - (y *x)))
      b = b - (eta * -y)
    }
  }
```

```
}

test_label = c()
# Evaluating on test data
for(i in 1:nrow(test_scaled_df))
{
  calculated_test = as.numeric((as.matrix(test_scaled_df[i,]) %*% t(a) ) + b )
  pred_test = ifelse(calculated_test >= 0, ">50K", "<=50K")
  test_label = c(test_label,pred_test)

}

# Export predicted label of test to txt file
write.table(test_label, file = "submission.txt", sep = "\n",
          row.names = FALSE)
```