Homework 2

Vishal Dalmiya (Dalmiya2); Himanshu Shah (Hs8); Deepak Nagarajan (deepakn2)

Feb 7, 2018

Problem Statement

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.

Write a program to train a support vector machine on this data using stochastic gradient descent. You should not use a package to train the classifier (that's the point), but your own code. You should ignore the id number, and use the continuous variables as a feature vector. You should scale these variables so that each has unit variance. You should 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. You should use at least 50 epochs of at least 300 steps each. In each epoch, you should separate out 50 training examples at random for evaluation (call this the set held out for the epoch). You should compute the accuracy of the current classifier on the set held out for the epoch every 30 steps.

Implementation

```r
library(klaR)
library(caret)
setwd('C:\\Personal\\AML\\HW2')

## DATA PREPARATION
train = read.csv('adult.data', header = FALSE)
test  = read.csv('adult.test', header = FALSE)
wdat = rbind(train, test)

# Extraction only the continuos features
data = wdat[, c(1, 3, 5, 11, 12, 13,15)]

bigx <- data.matrix(data[, -c(7)], rownames.force = NA)
bigy <- data.matrix(data[,c(7)], rownames.force = NA)

# Normalize each of the features
for (i in 1:ncol(bigx))
{
  bigx[, i] = (bigx[, i] - mean(bigx[, i])) / sd(bigx[, i])
}


# Convert the result to +1 & -1 for labeling purpose
for (i in 1:length(bigy))
```

```
{
  if ((bigy[i] == " <=50K") || (bigy[i] == " <=50K."))
    bigy[i] = -1
  else
    bigy[i] = 1
}


# Initilaize both "a" & "b" to 0s
a = as.matrix(rep(0, len = ncol(bigx)), nrow = 1)
b = 0

# First partition of dataset to 80-20
wtd = createDataPartition(y = bigy, p = .8, list = FALSE)

# train features
ntrbx <- bigx[wtd,]
# train labels
ntrby <- bigy[wtd]

# test/validation features
ntevabx <- bigx[-wtd,]
# test/validation labels
ntevaby <- bigy[-wtd]

# Second partition pf 20 into 10-10
wtd1 = createDataPartition(y = ntevaby, p = .5, list = FALSE)

# validation features
nvebx <- ntevabx[wtd1,]
# validation labels
nveby <- ntevaby[wtd1]


# test features
ntebx <- ntevabx[-wtd1,]
# test labels
nteby <- ntevaby[-wtd1]

# initialize lambdas, epochs, epoch steps and steps
lambda = c(exp(-11), exp(-9), exp(-7), exp(-5), exp(-3), exp(-2), exp(-1), 1)
epochs = 50
esteps = 20
steps  = 30

step_len = 1
inst     = 1

accuracy = matrix(0, nrow = length(lambda), ncol = 1000)
mag_a    = matrix(0, nrow = length(lambda), ncol = 1000)
accuracyVal = array(0, 4)
aTrain = matrix(0, nrow = length(lambda), ncol = 6)
bTrain = array(0, 4)
```

```r
j = 1

# iterate the training dataset for all lambdas
for(l in 1:length(lambda))
{
  lm = lambda[l]
  # Iterate through the number of epochs
  for (e in 1:epochs)
  {
    # initialize the step lenght
    step_len = 1 / (0.01 * e + 50)

    # Extract the 50 samples out and leave it aside (Held out set)
    tidx = sample(1:nrow(ntrbx), 50, replace = TRUE)

    for (s in 1:esteps)
    {
      for (i in 1:steps)
      {
        # Extract 1 sample from the training data set for evaluation
        inst = sample(1:nrow(ntrbx), 1, replace = TRUE)
        t_x = ntrbx[inst, ]

        # Compute the stochastic gradient
        if ((as.numeric(ntrby[inst]) * ((t_x %*% a) + b)) >= 1)
        {
          a = a - step_len * lm * a
        } else
        {
          a = a - step_len * (lm * a - as.numeric(ntrby[inst]) * t_x)
          b = b + step_len * as.numeric(ntrby[inst])
        }
      }

      j = (e - 1) * 20 + s

      # Compute the accuracy against the 50 samples (Held out set)
      res = (as.matrix(ntrbx[tidx, ]) %*% a  + b)
      idx = res > 0
      res[idx] = 1
      res[!idx] = -1
      gotright = res == bigy[tidx]
      accuracy[l, j] = sum(gotright) / (sum(gotright) + sum(!gotright))

      # Compute the magnitude of coefficient vector
      mag_a[l, j] = sqrt(t(a) %*% a)

    } # end of esteps

  } # end of epoch

  # compute the accuracy against the validation dataset
  res = (as.matrix(nvebx) %*% a  + b)
  idx = res > 0
  res[idx] = 1
  res[!idx] = -1
  gotright = res == nveby
  accuracyVal[l] = sum(gotright) / (sum(gotright) + sum(!gotright))
  aTrain[l,] = a
  bTrain[l] = b

}# end of lambda

# Display the value of lambdas and the corresponding accuracy on the validation
lambda
```

```
## [1] 0.0000167017 0.0001234098 0.0009118820 0.0067379470 0.0497870684
## [6] 0.1353352832 0.3678794412 1.0000000000
```

```r
accuracyVal
```

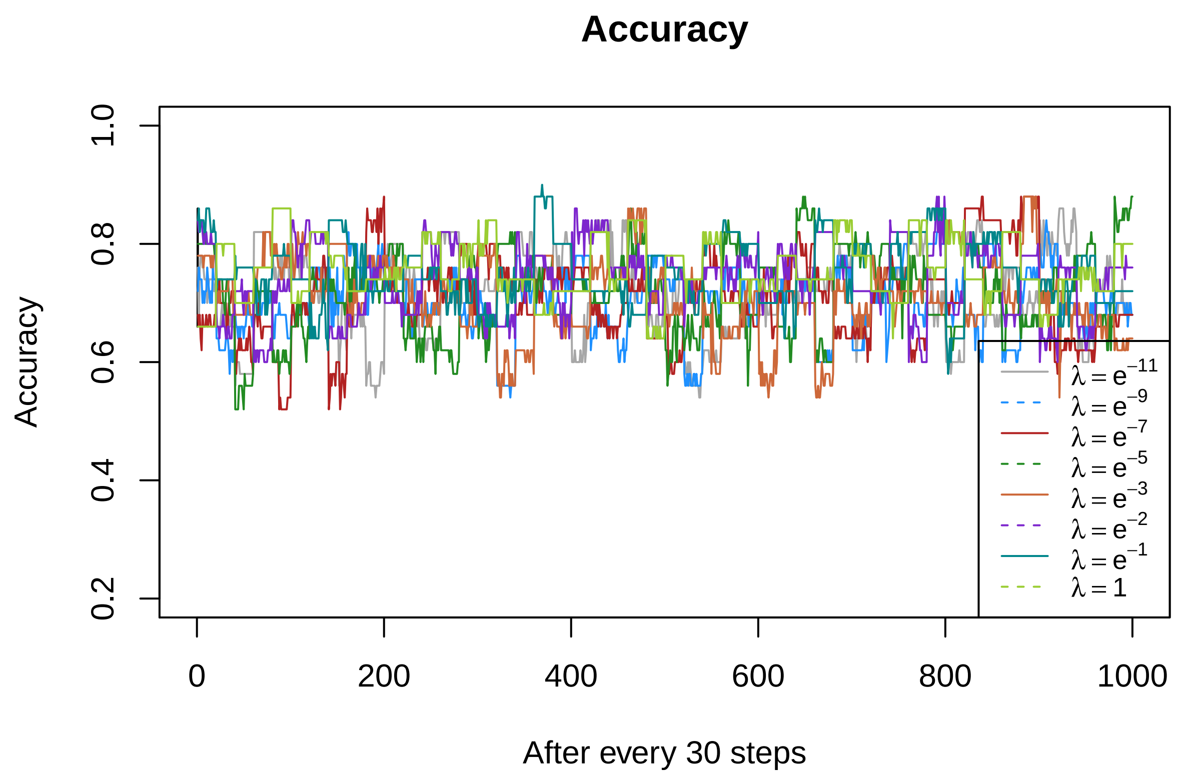
```
## [1] 0.8051177 0.8014330 0.8002047 0.7981576 0.7965200 0.7672467 0.7651996
## [8] 0.7658137
```

- We experimented with higher number of steps (600 instead of 300) per epoch to cover more data items, but we didnt see much difference in the accuracy. so we didn't try to cover the whole data set.

Results

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

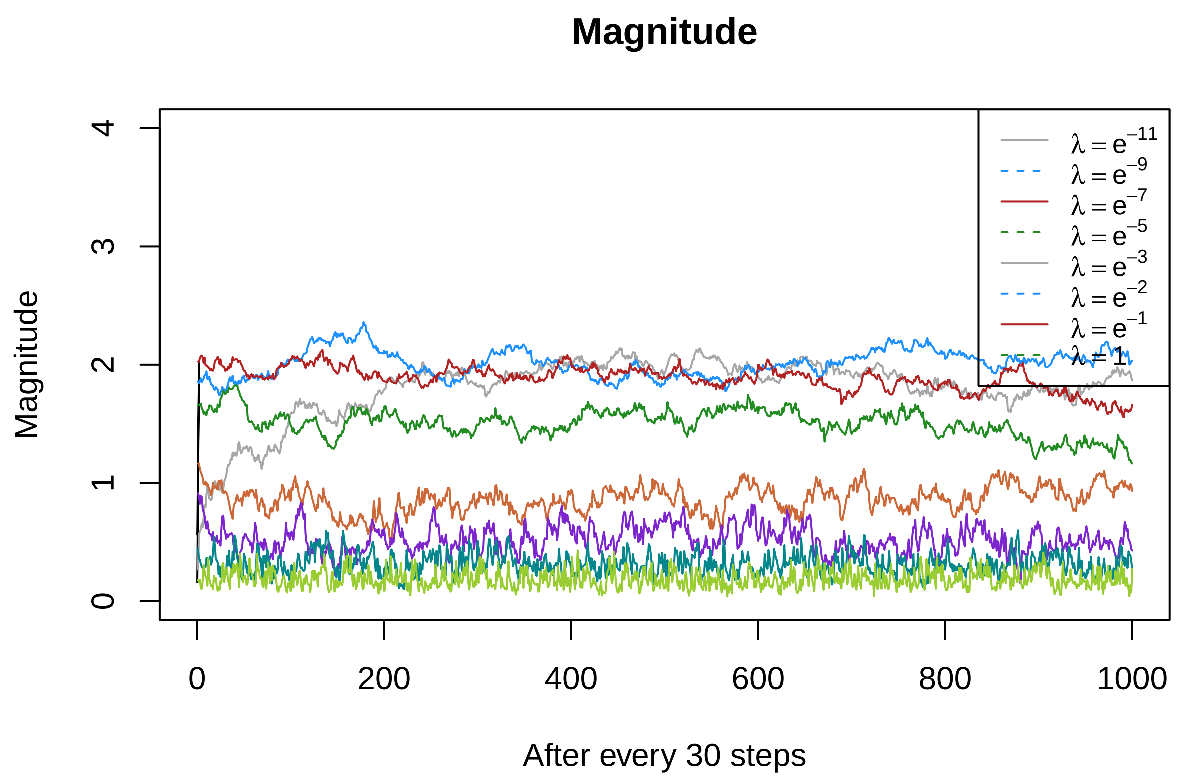
```r
# Plot Accuracy vs Epoch
plot(
  accuracy,
  main = "Accuracy",
  xlab = "After every 30 steps",
  ylab = "Accuracy",
  ylim = c(0.2, 1.0),
  xlim = c(0,1000),
  cex  = 1.5,
  lwd  = 1,
  pch  = 1,
  type = "l"
)


cl <- c("#A7A7A7",
        "dodgerblue",
        "firebrick",
        "forestgreen",
        "sienna3",
        "purple3",
        "turquoise4",
        "yellowgreen")

for (i in 1:8){
  lines(accuracy[i,],col = cl[i],type = 'l')
}

legend("bottomright", c(expression({lambda == e^-11}),expression({lambda == e^-9}),
                        expression({lambda == e^-7}),expression({lambda == e^-5}),
                        expression({lambda == e^-3}), expression({lambda == e^-2}),
                        expression({lambda == e^-1}), expression({lambda == 1})),
       col=cl[c(1,2,3,4,5,6,7,8)],lty=1:2, cex=0.8)
```



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

```
# Plot Size of a vs Epoch

plot(
  mag_a,
  main = "Magnitude",
  xlab = "After every 30 steps",
  ylab = "Magnitude",
  ylim = c(0, 4.0),
  xlim = c(0, 1000),
  cex  = 1.5,
  lwd  = 1,
  pch  = 1,
  type = "l"
)
```

```r
cl <- c("#A7A7A7",
        "dodgerblue",
        "firebrick",
        "forestgreen",
        "sienna3",
        "purple3",
        "turquoise4",
        "yellowgreen")

for (i in 1:8) {
  lines(mag_a[i, ], col = cl[i], type = 'l')
}

legend(
  "topright",
  legend = c(expression({lambda == e^-11}),expression({lambda == e^-9}),
             expression({lambda == e^-7}),expression({lambda == e^-5}),
             expression({lambda == e^-3}), expression({lambda == e^-2}),
             expression({lambda == e^-1}), expression({lambda == 1})),
  col = cl[c(1, 2, 3, 4)],
  lty = 1:2,
  cex = 0.8
)
```



Best value of the regularization constant based on 0.8051177, 0.801433, 0.8002047, 0.7981576, 0.79652, 0.7672467, 0.7651996, 0.7658137 is when

$$\lambda = e^-3$$

.

- We chose this as our regularization constant because of the comparatively high accuracy we have seen on the validation dataset for this lambda.
- Another reason we wanted to keep the lambda = eˆ-3, was to keep the magnitude of **"a"** lower to help with lower cost on un-seen data.

```
# Evaluating accuracy on the test dataset

epochs = 50
trainData = rbind(ntrbx, nvebx)
trainLabel = c(ntrby, nveby)
steps = round(length(trainLabel)/epochs,0)
lm = lambda[5]

#First calc a and b using whole train dataset (Including Train+Validation)
for(e in 1:epochs)
{
  # initialize the step lenght
  step_len = 1/(0.01 * e + 50 )

  # Extract the 50 samples out and leave it aside (Held out set)
  tidx = sample(1: nrow(trainData), 50, replace = TRUE)

  for(s in 1:steps)
  {

    # Compute the stochastic gradient
    inst = sample(1:nrow(trainData), 1, replace = TRUE)
    if ((as.numeric(trainLabel[inst]) * ((trainData[inst, ] %*% a) + b)) >= 1)
    {
      a = a - step_len * lm * a
    } else
    {
      a = a - step_len * (lm * a - as.numeric(trainLabel[inst]) * trainData[inst, ])
      b = b + step_len * as.numeric(trainLabel[inst])
    }


  } #end of steps
} # end of epoch

#Now Run this on Actual Test Data for lambda=e^-3 and the tuned parameters

res = (as.matrix(ntevabx[-wtd1, ]) %*% a  + b)
idx = res > 0
res[idx] = 1
res[!idx] = -1
gotright = res == ntevaby[-wtd1]
finalAccuracy = sum(gotright) / (sum(gotright) + sum(!gotright))
res = (as.matrix(ntevabx[-wtd1, ]) %*% a  + b)
idx = res > 0
res[idx] = 1
```

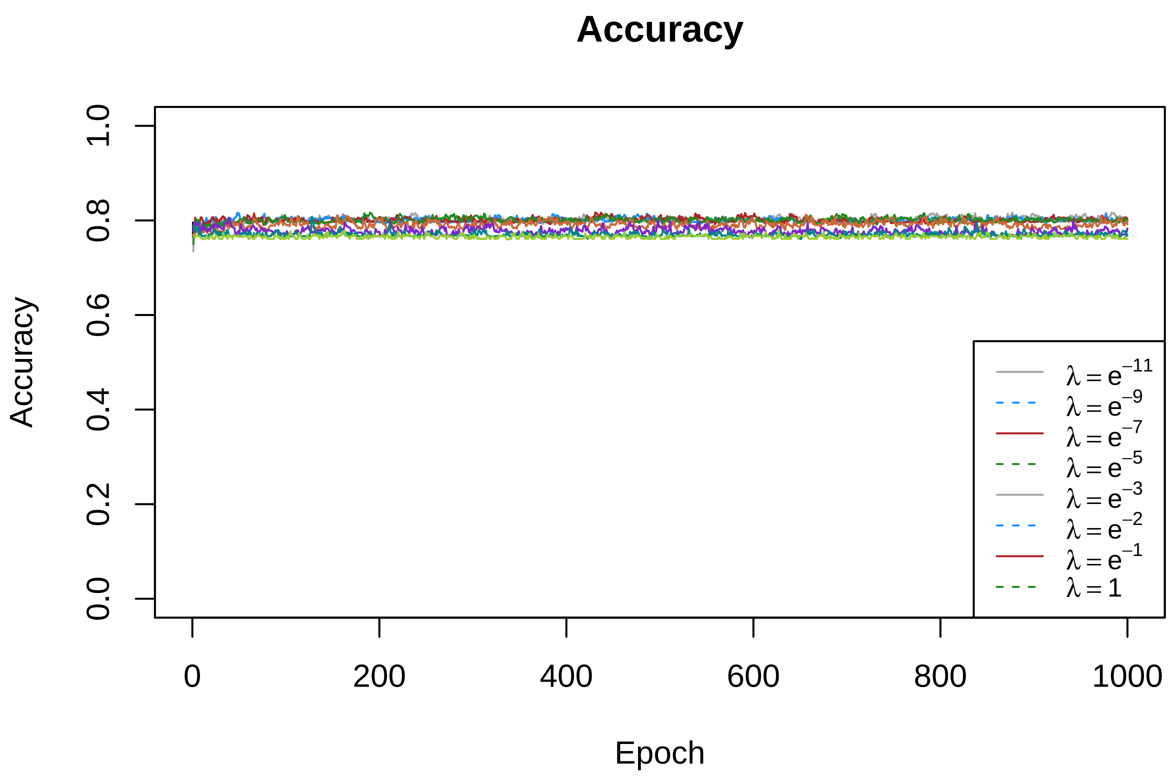
```
res[!idx] = -1
gotright = res == ntevaby[-wtd1]
accuracyFinal = sum(gotright) / (sum(gotright) + sum(!gotright))
```

- Accuracy of the best classifier with the chosen lamda, a & b, on the 10% test dataset data is **0.806881**

APPENDIX

- Here we tried a little experiment where the accuracy was based on test set to see if we can get a smoother curve as shown in the text book by the professor.

```
for(l in 1:length(lambda))
{
  lm = lambda[l]
  # Initilaize both "a" & "b" to 0s
  a = as.matrix(rep(0, len=ncol(bigx)), nrow = 1)
  b = 0

  # iterate the training dataset for all lambdas

  epochs = 50
  esteps = 20
  steps  = 30

  for(e in 1:epochs)
  {
    # initialize the step lenght
    step_len = 1/(0.01 * e + 50 )

    # Extract the 50 samples out and leave it aside (Held out set)

    for(s in 1:esteps)
    {

      for(i in 1:steps)
      {
        # Extract 1 sample from the training data set for evaluation
        inst = sample(1: nrow(ntrbx), 1, replace = TRUE)
        t_x = ntrbx[inst,]

        # Compute the stochastic gradient
        if((as.numeric(ntrby[inst]) * ((t_x %*% a) + b)) >= 1)
        {
          a = a - step_len * lm * a
        } else
        {
          a = a - step_len * (lm * a - as.numeric(ntrby[inst]) * t_x)
          b = b + step_len * as.numeric(ntrby[inst])
        }
      }

      j = (e-1)*20 + s
      # compute the accuracy against the validation dataset
      res = (as.matrix(ntebx) %*% a  + b)
      idx = res > 0
      res[idx] = 1
      res[!idx] = -1
      gotright = res == nteby
      accuracy[l,j] = sum(gotright) / (sum(gotright) + sum(!gotright))

      mag_a[l,j] = sqrt(t(a) %*% a)

    } # end of esteps

  } # end of epoch
}# end of lambda

# Plot Accuracy vs Epoch
plot(
  accuracy,
  main = "Accuracy",
  xlab = "Epoch",
  ylab = "Accuracy",
  ylim = c(0, 1.0),
  xlim = c(0,1000),
  cex  = 1.5,
  lwd  = 1,
  pch  = 1,
  type = "l"
)


cl <- c("#A7A7A7",
        "dodgerblue",
        "firebrick",
        "forestgreen","sienna3",
        "purple3",
        "turquoise4",
        "yellowgreen")

for (i in 1:8){
  lines(accuracy[i,],col = cl[i],type = 'l')
}

legend("bottomright", c(expression({lambda == e^-11}),expression({lambda == e^-9}),
                     expression({lambda == e^-7}),expression({lambda == e^-5}),
                     expression({lambda == e^-3}), expression({lambda == e^-2}),
                     expression({lambda == e^-1}), expression({lambda == 1})),
       col=cl[c(1,2,3,4)],lty=1:2, cex=0.8)
```



- The plot here looks very similar to the one professor has depicted in his book, this is because here we are running the accuracy on the entire test set, instead of the 50 samples in each step.