Assignment 3

ST340 Programming for Data Science

Released: Friday week 8, 2017-11-24; Deadline: ***11am on Friday week 10, 2017-12-08****.*

Instructions

- Work in groups of at least ONE and at most TWO.
- Specify your student numbers and names on your assignment.
- Any programming should be in R. Your report should be created using R Markdown and include any code you have written.
- Hand in the compiled assignment, clearly marked with "ST340 Assignment 3" to the Statistics department undergraduate office.

Student number(s):

Student name(s):

Q1

Here is a function that does gradient descent to find local minima:

```
gradient.descent <- function(f, df, x0, iterations=1000, alpha=0.2) {
  x<-x0
  for (i in 1:iterations) {
    cat(i,"/",iterations,": ",x," ",f(x),"\n")
    x<-x-alpha*df(x)
  }
  return(x)
}
```

Example:

```
f <-function(x) { sum(x^2) }
df<-function(x) { 2*x }
gradient.descent(f,df,c(10,20),10,0.2)
```

Q1a

> Write a *short* function that uses `gradient.descent` to find a local *maxima*. (For the purpose of this question, `gradient.descent` is a "black box". Don't worry about the printed output, just the return value matters.)

i.e.

```
gradient.ascent <- function(f, df, x0, iterations=1000, alpha=0.2) {
... use gradient.descent(...) here ...
}
f <-function(x) { (1+x^2)^(-1) }
df<-function(x) { -2*x*(1+x^2)^(-2) }
gradient.ascent(f,df,3,40,0.5)
```

Q1b

Consider the function $f : R^2 \to R$

```
f <- function(x) (x[1]-1)^2 + 100*(x[1]^2-x[2])^2
```

(1) Give a short mathematical proof that f has a a unique minima.
(2) Write a function df to calculate the gradient of f, i.e.
`df <- function(x) { ... use x[1] and x[2] ...}`

(3) Starting from the point `x0=c(3,4)`, try to find the minimum using gradient descent.
`gradient.descent(f,df,c(3,4), ... , ...)`

Q1c

Write a function to do gradient descent with momentum. Starting from the point `x0=c(3,4)`, use your function to the minimum of the function from part b.

Q2

Load the tiny MNIST dataset:

```
load("mnist.tiny.RData")
train.X=train.X/255
test.X=test.X/255
```

show some digits:

```
library(grid)
grid.raster(  array(aperm(array(train.X[1:50,],c(5,10,28,28)),c(4,1,3,2)),c(140,280)),
              interpolate=F)
```

Use 3-fold cross validation on the training set to compare SVMs with linear kernels, polynomial kernels and RBF kernels. i.e.

```
library(e1071)
set.seed(12345)
#We need to set seed so that the cross validation uses the same groups each time,
#otherwise changes in accuracy can be arbitrary due to changes in the groups used.

svm(train.X,train.labels,type="C-classification",kernel="linear",cross=3)$tot.accuracy


#INSERT CODE HERE ED
svm(train.X,train.labels,type="C-classification",kernel="poly",
    degree=2,coef=1,cross=3)$tot.accuracy

#Our gridsearchRBF function below gives us these values for radial kernal
svm(train.X,train.labels,type="C-classification",kernel="radial",
    gamma =exp(-4),cost=exp(3),cross=3)$tot.accuracy
```

etc, etc.

For the RBF kernels, write a grid search function that takes two lists, `log.C.range` and `log.gamma.range`, and for each pair (lc,lg) of entries in the pair of lists attempts cross-validation with parameters `cost =`

exp(lc) and gamma=exp(lg). Once you have found the model with the best cross-validation error, train it on the full training set and then test it on the test set.

```
#start with length(lc) and length(lg) as 3.

gridsearchRBF <- function(lc, lg){
  gridmatrix <- matrix(ncol = length(lc), nrow = length(lg))
  for(i in 1:length(lg)){
    for(j in 1:length(lc)){
      #start by filling the gridmatrix with initial accuracies
      gridmatrix[i,j] <- svm(train.X,train.labels,type="C",kernel="radial",
                             gamma=exp(lg[i]),cost=exp(lc[j]), cross = 3)$tot.accuracy
    }
  }
  griddy <- gridmatrix
  #Now find row and column of maximum value
 indexes <- which(griddy == max(griddy), arr.ind = TRUE)


  while((indexes[1,1] == 1 || indexes[1,1] == length(lg) || indexes[1,2] == 1 || indexes[1,2] == length
  new.col=c()
  new.row=c()
 print(griddy)
  #indexes[1,1] represents row
  #indexes[1,2] represents column
  if(indexes[1,2] == 1){
    if(indexes[1,1] == 1){
      #add another top row and left column
      lc <- c(lc[1]-1, lc)
      for(i in 1:length(lg)){
      new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                             gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy
      }
      griddy <- cbind(new.col, griddy)
      lg <- c(lg[1]-1, lg)
      for(i in 1:length(lc)){
      new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
                             gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
      }
      griddy <- rbind(new.row, griddy)


    }
    else if(indexes[1,1] == length(lg)){
      #add another bottom row and left column
      lc <- c(lc[1]-1, lc)
      for(i in 1:length(lg)){
      new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                             gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy
      }
      griddy <- cbind(new.col, griddy)
      lg <- c(lg, lg[length(lg)]+1)
      for(i in 1:length(lc)){
      new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
```

```
                              gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
    }
    griddy <- rbind(griddy, new.row)


  }
  else{
    #just add another left column
    lc <- c(lc[1]-1, lc)
    for(i in 1:length(lg)){
    new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                              gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy

    }
    griddy <- cbind(new.col, griddy)
  }

}
else if(indexes[1,2] == length(lc)){
  if(indexes[1,1] == 1){
    #add another top row and right column
    lc <- c(lc, lc[length(lc)]+1)
    for(i in 1:length(lg)){
    new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                              gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy
    }
    griddy <- cbind(griddy, new.col)
    lg <- c(lg[1]-1, lg)
    for(i in 1:length(lc)){
    new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
                              gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
    }
    griddy <- rbind(new.row, griddy)

  }
  else if(indexes[1,1] == length(lg)){
    #add another bottom row and right column
          lc <- c(lc, lc[length(lc)]+1)
    for(i in 1:length(lg)){
    new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                              gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy
    }
    griddy <- cbind(griddy,new.col)
    lg <- c(lg, lg[length(lg)]+1)
    for(i in 1:length(lc)){
    new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
                              gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
    }
    griddy <- rbind(griddy, new.row)

  }
  else{
    #just add another right column
```

```r
    lc <- c(lc, lc[length(lc)]+1)
    for(i in 1:length(lg)){
    new.col[i] <- svm(train.X,train.labels,type="C",kernel="radial",
                           gamma=exp(lg[i]),cost=exp(lc[1]), cross = 3)$tot.accuracy
    }
    griddy <- cbind(griddy, new.col)
  }
 }
 #Only possible combinations left are top row but not a corner
 #or bottom row but not a corner
 else if(indexes[1,1] == 1){
   #add another top row
   for(i in 1:length(lc)){
    new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
                           gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
    }
    griddy <- rbind(new.row, griddy)

   }
 else if(indexes[1,1]==length(lg)){
   #add another bottom row
   lg <- c(lg, lg[length(lg)]+1)
    for(i in 1:length(lc)){
    new.row[i] <-c(svm(train.X,train.labels,type="C",kernel="radial",
                           gamma=exp(lg[1]),cost=exp(lc[i]), cross = 3)$tot.accuracy)
    }
    griddy <- rbind(griddy, new.row)

 }
indexes <- which(griddy == max(griddy), arr.ind = TRUE)

#We need to consider if the max value is included in the grid more than once. This if statement
#will test if the max value appears more than once. If all are in the middle, it will delete
#all but one of them, then loop round again and we have our max.
#If some or all are on the edge, it will delete all but one that is on the edge, then loop round
#again, on this loop, if the newly added row or column is still smaller than all the maxes,
#it will pick up the maxes which we previously deleted and loop back round. (The max used initially wi

#Note we don't know how frequently this happens but it did happen a couple of times to us

if(nrow(indexes)>1){
  for(j in 1:nrow(indexes)){
    if((indexes[j,1] == 1 || indexes[j,1] == length(lg) || indexes[j,2] == 1 || indexes[j,2] == length
  indexes=indexes[j,]
  print(indexes)
  break
  }
}
}
 }
   gammavalue <- (lg[indexes[1,1]])
   costvalue <- (lc[indexes[1,2]])
   parameters <- c(gammavalue, costvalue)
```

```
    #will return first the log parameters and then the matrix it has produced.
  return(list(parameters,griddy))
}



set.seed(12345)
#After trying a few different vectors of lg and lc we found these values give us
#a high enough accuracy.
lg3 <- c(-3, -4, -5)
lc3 <- c(5, 4, 3)

griddy <- gridsearchRBF(lc3,lg3)

#We obtain that the best values are log(gamma)=-4, log(cost)=3
```

[[1]][1] -4 3

[[2]] new.col [1,] 88.8 89.3 88.6 87.1 [2,] 90.8 90.5 92.0 90.5 [3,] 88.8 89.7 89.6 90.2

```
set.seed(12345)

line <- svm(train.X,train.labels,type="C-classification",kernel="linear",cross=3)

pol <- svm(train.X,train.labels,type="C-classification",kernel="poly",
    degree=3,coef=2,cross=3)

radial <- svm(train.X,train.labels,type="C-classification",kernel="radial",
    gamma =exp(-4),cost=exp(3),cross=3)



mean(predict(line,train.X)==train.labels) #1
mean(predict(line,test.X)==test.labels) #0.866


mean(predict(pol,train.X)==train.labels) #0.95
mean(predict(pol,test.X)==test.labels) #0.888


mean(predict(radial,train.X)==train.labels) #1
mean(predict(radial,test.X)==test.labels) #0.916
```

A mean prediction of 1 in the training set indicates that the svm can correctly classify all numbers (images) in the training set. Both linear kernal and radial kernal achieve this, however polynomial kernal failed to achieve this, instead only getting 0.95.
Similarly for testing data, a mean value of 1 indicates everything was correctly classified. Linear, polynomial and radial SVM's achieved 0.866, 0.888 and 0.916 respectively. Hence, from the test data we can say that radial kernal is the best at classifying the data, then polynomioal, and then linear.