Multivariate Logistic Regression Implementation From Scratch

Author : Kisha Taylor

```
# Project#1 - Machine Learning Course
# March 5, 2018
# Prepared By : Kisha Taylor

# Multivariate Logistic Regression Implementation from Scratch

# Exploratory Analysis
#getwd()


#install.packages('Rcpp', dependencies = TRUE)
#install.packages('caret', dependencies = TRUE)
#install.packages('stringr')
library(stringr)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
#remove.packages("ggplot2")
#install.packages("ggplot2")
library(ggplot2)


#install.packages("xlsx")
library(xlsx)
```

```
## Loading required package: rJava
```

```
## Loading required package: xlsxjars
```

```
## Note required packages must be installed and loaded before running code
#Mice Protein Expression Data set found here:
#https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression


MiceDsOrig <- read.xlsx("C:/Users/Kisha/Downloads/Data_Cortex_Nuclear.xls",sheetName='Hoja1')


dim(MiceDsOrig)
```

```
## [1] 1080   82
```

```
class(MiceDsOrig)
```

```
## [1] "data.frame"
```

```
colnames(MiceDsOrig)[82]
```

```
## [1] "class"
```

```
# Using all but 1st column reflecting ID ( neither relevant independent or dependent
variable)  MDs <- MiceDsOrig[,-1]

# Converting all categorical variables to integers in separate columns
int_factors <- dummyVars(" ~ .",data = MDs)
newMiceDsOrig <- data.frame(predict(int_factors, MDs))
newMiceDsOrig[1:3,80:91]
```

```
##   Treatment.Memantine Treatment.Saline Behavior.C.S Behavior.S.C
## 1                   1                0            1            0
## 2                   1                0            1            0
## 3                   1                0            1            0
##   class.c.CS.m class.c.CS.s class.c.SC.m class.c.SC.s class.t.CS.m
## 1            1            0            0            0            0
## 2            1            0            0            0            0
## 3            1            0            0            0            0
##   class.t.CS.s class.t.SC.m class.t.SC.s
## 1            0            0            0
## 2            0            0            0
## 3            0            0            0
```

```
#Re-combining  1st column with newly converted columns
newMiceDsOrig <- cbind(MiceDsOrig[,1],newMiceDsOrig)

# re-instating original column name
colnames(newMiceDsOrig)[1] <- colnames(MiceDsOrig)[1]

################ Extracting selected atributes in new DS   #######################

############    Cleaning data - replacing NAs with mean

cleanDset <- function(Dset){
  replaceNAs <- function(x){
    x[which(is.na(x))] <- mean(x,na.rm=TRUE)
    return(x)
  }
  # identifying only the columns with numeric values
  #Code Ref:https://stackoverflow.com/questions/27475818/how-to-find-all-numeric-columns-in-data
  numerics <- colnames(Dset)[which(sapply(Dset,is.numeric))]

  #replaces NAs in numeric columns as specified
  Dset[,numerics] <- apply(Dset[,numerics],2,replaceNAs)
  return(Dset)
}


###################################################
# Getting newly cleaned data set
cl_Dset <- cleanDset(newMiceDsOrig)

########################################################
# Excluding 1st non-numeric column
newMice  <- cl_Dset[,-1]


###########################################################################
#####              ANALYSIS SEction

############################################################################


################ Preparing Data for use in Algorithm   #######################

FinalAttrOnly <- c("pPKCAB_N","APP_N","SOD1_N","Ubiquitin_N","CaNA_N")

FinalAllAttri <- c(FinalAttrOnly,colnames(newMice)[84:91])
N <- nrow(newMice)
# Add xo=1 column ( bias value)
x0 <- rep(1,N)
length(x0)
```

```
## [1] 1080
```

```
# For convenience
MDs <- cbind(x0,newMice[FinalAllAttri])
MDs <- MDs[sample(nrow(MDs)),]

# Splittng test and training data (30% vs. 70% of orig. data set size -respectively)
MaxTR <- 0.7*N
TrMDs <- MDs[1:MaxTR,]
TestMDs <- MDs[(MaxTR+1):N,]

length(colnames(TrMDs))
```

```
## [1] 14
```

```
######################### End of Data Prep. for Algo. #################




#############################################################################

#######        Implement "Logistic Discrimination"  #######################

##############################################################################

#Learning factor etha assigned vey small value,
# controls the step size for how the weights will
#vary for the gradient descent algorithm.


d <- ncol(TrMDs) -9 # Calculates # dimensions of X : 5 excluding bias unit
K <- 8

# spliting dataset into inputs( including bias variable) and outputs "r"
TrX <-  TrMDs[,(1:(d+1))] # 1st d+1 columns including bias unit
TrR <-  TrMDs[,((d+2):(d+1+K))] # all output columnsstarting after input values

#### Gradient Descent Algorithm #####

GradientDescent <- function(X,R,etha,Max_iter) {

  UpdateWeights <- function(w,X,R,etha){
    chgw <- matrix(rep(0,K*(d+1)),nrow=K) # initializing change in weights to zero
    N <- nrow(X)
    y <-  c()
    o <- c()

    for (t in 1:N){
      x <- as.matrix(X[t,])
      r <- as.matrix(R[t,])
      o <- w%*%x
      #Preprocess to avoid "overflow" by subtracting all o values from the maximum o value
      o <- o-o[which.max(o)]
      y <- as.matrix(exp(o)/sum(exp(o)))
      error <- r-y
      chgw <- error%*%t(x)
      w <- w + etha*chgw
    }# "t" for Loop
    return(w)
  } #end function UpdateWeights

  iter <- 0
  set.seed(105)
  wRandom <- c(runif((d+1)*K,-0.01,0.01))
  w <- matrix(wRandom,nrow=K)
  X <- as.matrix(X)
  R <- as.matrix(R)

  while (iter < Max_iter) {
    w <- UpdateWeights(w,X,R,etha)
    iter <- iter +1
  }# end while Loop

  message(sprintf("iter count: %s\n", iter))
  message(sprintf("etha: %s\n", etha))
  return(w)
}# end Gradient Descent Algorithm




ClassifyPts <- function(Testset,w){

  Ntest <- nrow(Testset)
  Ncol  <- ncol(Testset)
  TestX <-  Testset[,(1:(d+1))]
  TestR <-  Testset[,((d+2):Ncol)]
  y2 <-c()
  Error <- c()
  o <- c()
  SumErr <- 0
  pred <- matrix(rep(0,K*Ntest),ncol=K)
  colnames(pred) <- colnames(TestR)
  for (t in 1:Ntest){
    o <- w%*%t(as.matrix(TestX[t,]))  # Note input attributes start at index =5 up to 10 ( 1-4 are output,rt, labels)
```

```
    #Preprocess to avoid "overflow" by subtracting all o values from the maximum o value
    o <- o-o[which.max(o)]

    y2 <- exp(o)/sum(exp(o))
    maxi <- which.max(y2) # find max yi
    pred[t,maxi] <- 1
    Error[t] <- abs(TestR[t,maxi]-pred[t,maxi])
  }# end for loop (t in 1:Ntest)
  PercentError <- (sum(Error)/Ntest)*100
  ErrorCal <- data.frame(cbind(Error,TestR,pred))
  return(list(PercentError,n_etha,ErrorCal))
} # End of function, ClassifyPts




##################################################################
################  My Results for DataSet#2 - Logistic Discrimnation
##################################################################
n_etha = 0.0001

system.time(myW1700_GDS1t <- GradientDescent(X=TrX,R=TrR,etha=n_etha,Max_iter=1700))#Error =
```

```
## iter count: 1700
```

```
## etha: 1e-04
```

```
##    user  system elapsed
##   48.22    0.02   48.28
```

```
Error_newW1700_t <- ClassifyPts(Testset=TestMDs,w=myW1700_GDS1t)
Error_newW1700_t[[1]]
```

```
## [1] 53.08642
```

```r
#Results
#n_etha = 0.0001
#Error_newW50_t[[1]] # Error : 63.2716
#Error_newW500_t[[1]] # Error : 53.08642
#Error_newW1500_t[[1]] # Error : 48.45679
#Error_newW1700_t[[1]] # Error : 48.14815
#Error_newW1800_t[[1]] # Error : 48.14815

############################################################
###########Test results other metrics & Confusion Matrix
###########################################################
getConfusionMat <- function(PredDf,GrTr){
  K <- ncol(GrTr)
  rnum <- nrow(GrTr)
  confM <- matrix(rep(0,K*K),nrow=K)
  colnames(PredDf)<- str_replace(colnames(PredDf),".1","")
  colnames(PredDf)<- str_replace(colnames(PredDf),"class.","Pred_")
  colnames(confM) <- colnames(PredDf)
  rownames(confM) <- str_replace(colnames(PredDf),"Pred","GrTr")

  for (tcnt in 1:rnum){
    for (cP in 1: K){
      for (cr in 1:K){
        if ((PredDf[tcnt,cP]==1) && (GrTr[tcnt,cr]==1)){
          confM[cP,cr] <- confM[cP,cr] + 1
        }

      }
    }
  }
  return(confM)
}



ReportMetrics <- function(ConfMat){
  # Recall for a particular class measures fraction of examples belonging to a particular class that were predicted correctl
y.
  # Precision measures fraction of all prediction for a particular class that were correct.
  # Accuracy measures the fraction of all predictions that were correct.
  Recall <- c()
  Precision <- c()
  nrowConfMat <- nrow(ConfMat)
  ncolConfMat <- ncol(ConfMat)
  Precision <- c()
  if (nrowConfMat != ncolConfMat){
    return(print("Matrix must be N X N - double ck. dimensions"))
  }
  else {
    for (c in 1:ncolConfMat){
      Recall[c]    <- 100*ConfMat[c,c] / sum(ConfMat[c,])
      Precision[c] <- 100* ConfMat[c,c] / sum(ConfMat[,c])
    }
    if (sum(is.na(Recall)) >0){
      Recall[which(is.na(Recall))] <- 0
    }
    if (sum(is.na(Precision)) >0){
      Precision[which(is.na(Precision))] <- 0
    }
    AvgRecall    <- mean(Recall)
    AvgPrecision <- mean(Precision)
    AccCal <- 100*(sum(diag(ConfMat)))/sum(ConfMat)
  }
  print(paste("Accuracy: ",round(AccCal,2),"%"))
  for (c in 1:ncolConfMat){
    print(paste("Precision for class",gsub("GrTr_","",rownames(ConfMat)[c]),":",Precision[c],"%"))
  }
  for (c in 1:ncolConfMat){
    print(paste("Recall for class",gsub("GrTr_","",rownames(ConfMat)[c]),":",Recall[c],"%"))
  }
  return(list(AccCal,Recall,Precision,AvgRecall,AvgPrecision ))
}


########################################################################
### Reporting Test results ( Metrics)
########################################################################

Predictions <- (Error_newW1700_t[[3]][,c(10:17)])
GroundTruth<- Error_newW1700_t[[3]][,c(2:9)]
```

```
ConfusionMatrix <- getConfusionMat(Predictions ,GroundTruth)
Metrics <- ReportMetrics(ConfusionMatrix)
```

```
## [1] "Accuracy:  46.91 %"
## [1] "Precision for class c.CS.m : 45.6521739130435 %"
## [1] "Precision for class c.CS.s : 72.972972972973 %"
## [1] "Precision for class c.SC.m : 100 %"
## [1] "Precision for class c.SC.s : 55.8139534883721 %"
## [1] "Precision for class t.CS.m : 52.3809523809524 %"
## [1] "Precision for class t.CS.s : 0 %"
## [1] "Precision for class t.SC.m : 12.1951219512195 %"
## [1] "Precision for class t.SC.s : 43.75 %"
## [1] "Recall for class c.CS.m : 42.8571428571429 %"
## [1] "Recall for class c.CS.s : 38.5714285714286 %"
## [1] "Recall for class c.SC.m : 38.0952380952381 %"
## [1] "Recall for class c.SC.s : 50 %"
## [1] "Recall for class t.CS.m : 59.4594594594595 %"
## [1] "Recall for class t.CS.s : 0 %"
## [1] "Recall for class t.SC.m : 41.6666666666667 %"
## [1] "Recall for class t.SC.s : 87.5 %"
```

```
Accuracy<- Metrics[[1]]
Recall<- Metrics[[2]]
Precision <- Metrics[[3]]
AvgRecall <- Metrics[[4]]
AvgPrecision <- Metrics[[5]]
```