Multilayer Perceptron Classifier Implementation from Scratch

(Basic feed-forward Artificial Neural Network-ANN)

Author : Kisha Taylor

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

# Classification using Multilayer Perceptron
# -BackPropagation Algorithm 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
```

```
#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

Testx<-TestMDs[,(1:(d+1))]
Testr<-TestMDs[,((d+2):(d+1+K))]

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

##2 / Implement Multilayer backpropagation algorithm and predict

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

Pred_y <-function(w,v,x,r){ # Predicts y given w & v weights on test set
  xrows <- nrow(x)
  zpred<- rep(1,H+1)
  pred <- matrix(rep(0,K*xrows),ncol=K)
  colnames(pred) <- colnames(r)[1:K]
  Error <- c()
  Esty<- matrix(rep(0,K*xrows),nrow=xrows)
  opred <- matrix(rep(0,K*xrows),nrow=xrows)

  for (t in 1:xrows){
    zpred[-1]<-sigmoid((t(as.matrix(w))%*%as.matrix(t(x[t,]))))
    opred[t,] <- (t(as.matrix(v))%*%as.matrix(zpred))
    Esty[t,] <- exp(opred[t,])/sum(exp(opred[t,]))

    max  <- max(Esty[t,])
    maxi <- which.max(Esty[t,])
    pred[t,maxi] <- 1
    Error[t] <- abs(r[t,maxi]-pred[t,maxi])
  }# end for loop (t in 1:xrows)

  PercentError <- (sum(Error)/xrows)*100
  ErrorCal <- data.frame(cbind(Error,pred))
  return(list(PercentError,ErrorCal))
}# end Pred_y function



sigmoid <- function(f){
  return(1/(1+exp(-f)))
}

UpdateWeights <- function(t,v,w,x,r,etha){
  y <- c()
  o <- c()
  z <- rep(1,H+1)
  chngv <- matrix(rep(0,(H+1)*K),ncol=K)
  chngw <- matrix(rep(0,(d+1)*H),ncol=H)

  z[2:(H+1)] <- sigmoid(t(as.matrix(w))%*%t(as.matrix(x[t,])))
  o <- t(as.matrix(v))%*%z
  y <- exp(o)/sum(exp(o))
  chngv <- t(etha*t(as.matrix(r[t,] - y))%*%t(as.matrix(z)))
  err <- (as.matrix((r[t,] - y)))%*%(as.matrix(t(v[(2:(H+1)),])))
  chngw <-  t(etha*((t(err)%*%t(as.matrix(z[2:(H+1)])))%*%((as.matrix(1-z[2:(H+1)]))%*%as.matrix(x[t,]))))
  v <- v + chngv
  w <- w + chngw
  return(list(w,v)) ## returns list with updated weights w and v.
}# end  Updateweights


H <- 3 # No. of  dimensions in hidden space
K <- 8 # No. of dependent variables (note: this is a univariate regression problem since K=1)
d <- 5 # No. of dimensions of X input variable (independent variable,excludes bias input unit)

BackProp <- function(x,r,etha,Max_iter) {   #BackProp(dfTr,etha=n_etha,Max_iter=100)

  # initialize weights in x space and z hidden space
  set.seed(500)
  v <-  matrix(runif(((H+1)*K),-0.01,0.01),ncol=K)
  set.seed(501)
```

```
  w <-  matrix(runif(((d+1)*H),-0.01,0.01),ncol=H)

  iter <- 0
  NTrRows <- nrow(x)
  while (iter < Max_iter) {
    for (t in 1:NTrRows) {
      newWV <- UpdateWeights(t,v,w,x,r,etha)
      w <- as.matrix(newWV[[1]]) # weights updated for W (in list)
      v <- as.matrix(newWV[[2]]) # weights updated for V (in list)
    }#
    iter <- iter +1
    #print(iter)
  }# end while loop

  return(list(w,v))
}# end function BackProp





##################################################################
###############  My Results for DataSet#2 - BackPropagation - MLP
##################################################################
n_etha = 0.01
Max_iter <- 235
system.time(ResultBackProp <- BackProp(x=TrX,r=TrR,etha=n_etha,Max_iter))
```

```
##    user  system elapsed
##  551.86    0.39  562.16
```

```
w<-ResultBackProp[[1]]
v<-ResultBackProp[[2]]
Est_y <- Pred_y(w,v,x=Testx,r=Testr)
ErrorCal <- Est_y[[1]]
ErrorCal
```

```
## [1] 58.95062
```

```
#Results
#n_etha = 0.01
#ResultBackProp50 # Error: 62.03704
#ResultBackProp100 # Error: 55.55556
#ResultBackProp200 # Error: 48.45679 time elapsed: 504.41
#ResultBackProp220 # Error: 48.14815
#ResultBackProp230 # Error: 48.14815
#ResultBackProp250 # Error: 48.45679
#ResultBackProp300 # Error: 48.76543

#############################################################
###########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),"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 <- Est_y[[2]][,-1]
GroundTruth <- Testr
nrow(Predictions)
```

```
## [1] 324
```

```
nrow(GroundTruth)
```

```
## [1] 324
```

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

```
## [1] "Accuracy:  41.05 %"
## [1] "Precision for class c.CS.m : 96.969696969697 %"
## [1] "Precision for class c.CS.s : 2.7027027027027 %"
## [1] "Precision for class c.SC.m : 21.2765957446809 %"
## [1] "Precision for class c.SC.s : 61.3636363636364 %"
## [1] "Precision for class t.CS.m : 0 %"
## [1] "Precision for class t.CS.s : 0 %"
## [1] "Precision for class t.SC.m : 80 %"
## [1] "Precision for class t.SC.s : 81.3953488372093 %"
## [1] "Recall for class c.CS.m : 21.0526315789474 %"
## [1] "Recall for class c.CS.s : 100 %"
## [1] "Recall for class c.SC.m : 71.4285714285714 %"
## [1] "Recall for class c.SC.s : 67.5 %"
## [1] "Recall for class t.CS.m : 0 %"
## [1] "Recall for class t.CS.s : 0 %"
## [1] "Recall for class t.SC.m : 40.5797101449275 %"
## [1] "Recall for class t.SC.s : 76.0869565217391 %"
```

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