HMM-BaumWelsh Implementation from scratch

Author : Kisha Taylor

```
## Project#2 - Data Mining
#  March 27, 2018
#  Prepared by: Kisha Taylor

# HMM (Hidden Markov MOdel Implementation)

#OccTR <- read.table("C:/Users/Kisha/Downloads/occupancy_data/datatraining.txt",header=TRUE, sep
 = ",")
#OccTest <- read.table("C:/Users/Kisha/Downloads/occupancy_data/datatest.txt",header=TRUE, sep =
 ",")

#OccTR <- read.table("D:/Users/kisha.taylor/Documents/datatraining.txt",header=TRUE, sep = ",")
#OccTest <- read.table("D:/Users/kisha.taylor/Documents/datatest.txt",header=TRUE, sep = ",")




#Datasets
#Occupancy Detection Data Set: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

#Analysis
#1. For both CO2 and the attribute you have chosen in the exploratory analysis

#a. Apply the Baum-Welch algorithm on the training set to learn the HMM model ?? = (A,B,p)
#b. Once you have learned the HMM model, using the Viterbi algorithm, generate the occupancy
#   sequence as your prediction results with your test set. Discuss its performance.

# Our Hidden variable is Occupancy and we are using CO2 and Light as our observation variables


getwd()
```

```
## [1] "D:/Users/kisha.taylor/Documents"
```

```
OccTR <- read.table("D:/Users/kisha.taylor/Documents/datatraining.txt",header=TRUE, sep = ",")
OccTest <- read.table("D:/Users/kisha.taylor/Documents/datatest.txt",header=TRUE, sep = ",")

#OccTR <- read.table("C:/Users/Kisha/Downloads/occupancy_data/datatraining.txt",header=TRUE, sep
 = ",")
#OccTest <- read.table("C:/Users/Kisha/Downloads/occupancy_data/datatest.txt",header=TRUE, sep =
 ",")
#OccTR <- OccTR[1:2600,]


class(OccTR)
```

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

```
summary(OccTR)
```

```
##                    date       Temperature       Humidity    
##  2015-02-04 17:51:00:   1   Min.   :19.00   Min.   :16.75  
##  2015-02-04 17:51:59:   1   1st Qu.:19.70   1st Qu.:20.20  
##  2015-02-04 17:53:00:   1   Median :20.39   Median :26.22  
##  2015-02-04 17:54:00:   1   Mean   :20.62   Mean   :25.73  
##  2015-02-04 17:55:00:   1   3rd Qu.:21.39   3rd Qu.:30.53  
##  2015-02-04 17:55:59:   1   Max.   :23.18   Max.   :39.12  
##  (Other)            :8137                                  
##      Light             CO2         HumidityRatio       Occupancy     
##  Min.   :   0.0   Min.   : 412.8   Min.   :0.002674   Min.   :0.0000  
##  1st Qu.:   0.0   1st Qu.: 439.0   1st Qu.:0.003078   1st Qu.:0.0000  
##  Median :   0.0   Median : 453.5   Median :0.003801   Median :0.0000  
##  Mean   : 119.5   Mean   : 606.5   Mean   :0.003863   Mean   :0.2123  
##  3rd Qu.: 256.4   3rd Qu.: 638.8   3rd Qu.:0.004352   3rd Qu.:0.0000  
##  Max.   :1546.3   Max.   :2028.5   Max.   :0.006476   Max.   :1.0000  
## 
```

```
colnames(OccTR)
```

```
## [1] "date"          "Temperature"   "Humidity"      "Light"        
## [5] "CO2"           "HumidityRatio" "Occupancy"
```

```
class(OccTR)
```

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

```
head(OccTR)
```

```
##                   date Temperature Humidity Light    CO2 HumidityRatio
## 1 2015-02-04 17:51:00       23.18  27.2720 426.0 721.25   0.004792988
## 2 2015-02-04 17:51:59       23.15  27.2675 429.5 714.00   0.004783441
## 3 2015-02-04 17:53:00       23.15  27.2450 426.0 713.50   0.004779464
## 4 2015-02-04 17:54:00       23.15  27.2000 426.0 708.25   0.004771509
## 5 2015-02-04 17:55:00       23.10  27.2000 426.0 704.50   0.004756993
## 6 2015-02-04 17:55:59       23.10  27.2000 419.0 701.00   0.004756993
##   Occupancy
## 1         1
## 2         1
## 3         1
## 4         1
## 5         1
## 6         1
```

```
class(OccTR[,4])
```

```
## [1] "numeric"
```

```
#######################################################################
# Pre-processing steps -Discretization                      ###########
######################################################################

# Observation variable :CO2 (hidden variable is Occupancy).###########
# Discritizing variable CO2 by changing the values such
# than they range from 1 to No. of observation (integer value)

#install.packages("numbers")
#install.packages("stringr")
#install.packages("CHNOSZ") #Used in Baum-Welsh Algo. (ForwardBackward)
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 3.4.4
```

```
library(numbers)
```

```
## Warning: package 'numbers' was built under R version 3.4.4
```

```
library(CHNOSZ)
```

```
## Warning: package 'CHNOSZ' was built under R version 3.4.4
```

```
## CHNOSZ version 1.1.3 (2017-11-13)
```

```
## Please run data(thermo) to create the "thermo" object
```

```
class(OccTR$Light)
```

```
## [1] "numeric"
```

```
class(OccTR$Temperature)
```

```
## [1] "numeric"
```

```
class(OccTR$Humidity)
```

```
## [1] "numeric"
```

```
class(OccTR$HumidityRatio)
```

```
## [1] "numeric"
```

```
class(OccTR$date)
```

```
## [1] "factor"
```

```
max(OccTR$Light)
```

```
## [1] 1546.333
```

```
min(OccTR$Light)
```

```
## [1] 0
```

```
Discretize <- function(Dset,minv,maxv,nObs,coln){ # returns dset discretized on specified variab
le

  start <- minv
  end <- maxv
  ints <- (end-start)/nObs


  if (ints/round(ints,0) !=1){
    return(print("Diff. between minvalue (minv)& maxvalue(maxv) for variable(in coln) must be mu
ltiple of # Observations (nObs)"))
  }
  else {
    interval_upperlimit <- c()
    Ulimit <- c()
    newVals <- Dset[,coln]
    newDset <- cbind(Dset,newVals)
    nVcoln  <- ncol(newDset)
    colname_nV <- gsub(" ","",(paste("new",(colnames(Dset)[coln]))," ")
    colnames(newDset)[nVcoln] <- colname_nV

    cnt <-1
    for (cnt in 1:nObs){
      if (cnt ==1){
        newDset[which((newDset[,nVcoln]>=0) & (newDset[,nVcoln]<= (start+(ints*cnt)))),colname_n
V] <- cnt
        Ulimit[cnt] <-  (start+(ints*cnt))
      }
      else {
        newDset[which((newDset[,nVcoln]>(start+(ints*(cnt-1))) & (newDset[,nVcoln]<=(start+(ints
*cnt))) )),colname_nV] <- cnt
        Ulimit[cnt] <-  (start+(ints*cnt))
      }
    }
    return(newDset)
  }
}



#### Choose dimensions of emission matrix (M)
# Dimensions:  nrows( # hiddden sates)  =  2 &&
#             ncols (#Disinct obs. states/values) = 6




#### Preparing Data set for use ###################
#### Collecting rows in Dataset based on timestamps and desired time interval


getDset <- function(Dset,chngt){
  numrows <- floor((nrow(Dset))/chngt)
  getIndx <- c()
  for (cnt in 1:numrows){
    if (cnt==1){
      getIndx[cnt] <- 1
    } else {
      getIndx[cnt] <- 1+(chngt*(cnt-1))
    }
  }
  print(getIndx)
  return(Dset[getIndx,])
}


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

initAB <- function(N,M){
  # takes Hidden states(N), # of distinct Observation symbols

  # transition matrix
  set.seed(5)
  a<- matrix(runif(N*N,5,50),nrow=N,ncol=N)

  for (rowp in 1:N){
    sumarow <- sum(a[rowp,])
    for (colp in (1:N)){
      a[rowp,colp] <- (a[rowp,colp]/sumarow)
    }
  }

  # emission matrix: bj(m)


  set.seed(6)
  b<- matrix(runif(N*M,5,50),nrow=N,ncol=M)

  for (rowp in 1:N){
    sumarow <- sum(b[rowp,])
    for (colp in (1:M)){
      b[rowp,colp] <- (b[rowp,colp]/sumarow)
    }
  }

  # initialization vector
  g <- c()
  set.seed(7)
  g[1:N]<- runif(N,5,50)
  sumg<- sum(g)
  for (cnt in 1:N){
    g[cnt] <- g[cnt]/sumg
  }


  colnames(a)<- as.character(qvals)
  rownames(a)<- as.character(qvals)
  ncol(b)
  colnames(b)<- as.character(Obsvals)
  rownames(b)<- as.character(qvals)


  return(list(a,b,g))
}


bCindx <- function(v,b){
  return(which(colnames(b)==v))
}




#####################################################
forward <- function(obsSeq,a,b,g){
  LT <- length(obsSeq)
  ct <- c()
  alf <- matrix(rep(0,N*LT),nrow=N)

  #initializing alpha for t=1
  v1<- obsSeq[1]

  alf[,1]<- (g*b[,bCindx(v1,b)])
  ct[1] <- 1/sum(alf[,1])
  alf[,1] <- alf[,1]*ct[1]

  if (LT >1){
    for (t in 2:LT){
      v1<- obsSeq[t]
      for (j in 1:N){
        alf[j,t] <- (((t(alf[,t-1]))%*%a[,j])*(b[j,bCindx(v1,b)]))
      }
      ct[t] <- 1/sum(alf[,t])
      alf[,t] <- alf[,t]*ct[t]
    }
  }
  return(list(alf,ct))
}



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

backward <- function(obsSeq,a,b,g){
  Tm <- length(obsSeq)
  ct <- c()
  Beta<- matrix(rep(0,N*Tm),nrow=N)
  #initializing alpha for t=1

  Beta[,Tm]<- 1
  #ct[Tm] <- 1/sum(Beta[,Tm])  #Normalization step
  if (Tm >1){
    for (t in ((Tm-1):1)){
      v1<- obsSeq[t+1]
      for (i in 1:N){
        Beta[i,t] <- sum((a[i,])*(b[,bCindx(v1,b)])*(Beta[,t+1]))
      }
    }
    ct[t] <- 1/sum((Beta[,t]))

    #Normalization step
    Beta[,t] <- (Beta[,t])*ct[t]
  }

  return(list(Beta,ct))

}


#############################
Obseqindx <- function(vk,Oseq){
  return(which(Oseq==vk))
}
#vt<- obsSeq[t]
#Obseqindx(vt,obsSeq)


bCindx <- function(v,b){

  return(which(colnames(b)==v))
}


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

##########    Baum-Welsh Algorithm
##########    (or ForwardBackward )

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


forwardbackward <- function(obsSeq,maxiter){
  LenO <- length(obsSeq)

  #initialization
  a1 <- initAB(N,M)[[1]]
  b1 <- initAB(N,M)[[2]]
  g  <- initAB(N,M)[[3]]

  yt <- matrix(0,nrow=N,ncol=LenO)
  Gm <- array(0,dim=c(N,N,LenO))

  itercnt <- 0
  converge<-"False"
  while (converge=="False"){
    itercnt <- itercnt +1
    #E-Step (Expectation step)
    alf <- forward(obsSeq,a1,b1,g)[[1]]
    Beta <-backward(obsSeq,a1,b1,g)[[1]]

    for(t in 1:(LenO-1)){
      vtplus1<- obsSeq[t+1]
      for (i in 1:N){
        for (j in 1:N){
          numerGm <- (alf[i,t]*a1[i,j]*b1[j,bCindx(vtplus1,b1)]*Beta[j,t+1])
          sumk <- 0
          for (k in 1:N){
            suml <- 0
            for (l in 1:N){
              suml <- suml + (a1[i,l]*b1[l,bCindx(vtplus1,b1)]*Beta[l,t+1])
            }
            sumk <- sumk + ((alf[k,t])*suml)
          }
          denomGm <- sumk
          Gm[i,j,t] <- numerGm/denomGm
        }

      }
    }



    for (t in 1:LenO){
      yt[,t] <- (alf[,t]*Beta[,t])/sum(alf[,t]*Beta[,t])
    }


    #M-Step
    # a1 rep. estimated transition matrix a

    preva1 <- a1


    for (i in 1:nrow(a1)){
      denumTerm <- sum(yt[i,(1:(LenO-1))])

      for (j in 1:ncol(a1)){
        numTerm <- sum(Gm[i,j,(1:(LenO-1))])

        a1[i,j] <- numTerm/denumTerm

      }
    }

    prevb1 <- b1

    for (j in 1:nrow(b1)){
      for (vcnt in 1:ncol(b1)){
        vt <- colnames(b1)[vcnt]
        b1[j,vcnt] <-  (sum(yt[j,Obseqindx(vt,obsSeq)]))/sum(yt[j,])
      }
    }
    prevg <- g
    g <-  yt[,1]
    if ( ( (identical(preva1,a1)) && (identical(prevb1,b1))&& (identical(prevg,g)) | (itercnt==
 maxiter) ) ){
      converge<-"True"
    }
    #
  }#end While Loop

  print(converge)
  print(identical(preva1,a1))
  print(identical(prevb1,b1))
  print(identical(prevg,g))
  print(itercnt)
  return(list(preva1,a1,prevb1,b1,g))
}



##########################################################
#####      Viterbi Algorithm                      #########
##########################################################



Viterbi <- function(obsSeq,a1,b1,g1){
  LT <- length(obsSeq)
  v1<- obsSeq[1]

  theta <- matrix(rep(0,N*LT),nrow=N)
  theta[,1] <- (g1*b1[,bCindx(v1,b1)])
  trellis <- matrix(rep(0,N*LT),nrow=N)

  for (t in 2:LT){
    v1<- obsSeq[t]
    for (s in 1:N){
      max <- -5

      for (i in 1:N){
        if (max < ((theta[i,t-1])*(a1[i,s]))*(b1[s,bCindx(v1,b1)])){
          max <- (theta[i,t-1])*(a1[i,s])*(b1[s,bCindx(v1,b1)])
          maxi <- i
        }
      }

      theta[s,t] <- max
      trellis[s,t] <- maxi

    }
  }
  print(theta)
  print(trellis)
  maxbk <- -10
  for (s in 1:N){
    if (maxbk < trellis[s,t]){
      maxbk <- trellis[s,t]
      maxbki <- s
    }
  }
  p <- maxbki

  return(list(trellis,p))
}

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



unpack<- function(estbkPath,p){
  Lenpath <- ncol(estbkPath)
  newPath <- c()
  bkpath <- c()
  indxOfHS <- p

  for (k in Lenpath:1){
    newPath[k] <- indxOfHS
    indxOfHS<- estbkPath[indxOfHS,k]

  }
  newPath[which(newPath==2)] <- 0
  return(newPath)
}

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



###############################################
###Calc. Accuracy of model on Dset (Test set)
AccCalc <- function(MyestPathfromt1,Dset1,ColnHS){
  cntRight <- 0
  cntDs <- nrow(Dset1)
  for (cnt in 1:cntDs){
    if (MyestPathfromt1[cnt] ==Dset1[cnt,ColnHS]){
      cntRight <- cntRight + 1
    }
  }
  Accuracy <- 100*cntRight/cntDs
  return(Accuracy)
}
```

```
#### Training HMM model ("learning"/estimating parameters)
#### given observation sequence and N (# of unique hidden states)


# Pre-processing step
coln = which(colnames(OccTR)=="Light")
newOccTR <- Discretize(Dset=OccTR,minv=0,maxv=1800,nObs=6,coln)



chngt <- 30 #Example, every 10mins., every 59 sec approximates to 1 min when no time stamp for t
he exact min.

newDset1 <- getDset(newOccTR,chngt)
```

```
##   [1]    1   31   61   91  121  151  181  211  241  271  301  331  361  391
##  [15]  421  451  481  511  541  571  601  631  661  691  721  751  781  811
##  [29]  841  871  901  931  961  991 1021 1051 1081 1111 1141 1171 1201 1231
##  [43] 1261 1291 1321 1351 1381 1411 1441 1471 1501 1531 1561 1591 1621 1651
##  [57] 1681 1711 1741 1771 1801 1831 1861 1891 1921 1951 1981 2011 2041 2071
##  [71] 2101 2131 2161 2191 2221 2251 2281 2311 2341 2371 2401 2431 2461 2491
##  [85] 2521 2551 2581 2611 2641 2671 2701 2731 2761 2791 2821 2851 2881 2911
##  [99] 2941 2971 3001 3031 3061 3091 3121 3151 3181 3211 3241 3271 3301 3331
## [113] 3361 3391 3421 3451 3481 3511 3541 3571 3601 3631 3661 3691 3721 3751
## [127] 3781 3811 3841 3871 3901 3931 3961 3991 4021 4051 4081 4111 4141 4171
## [141] 4201 4231 4261 4291 4321 4351 4381 4411 4441 4471 4501 4531 4561 4591
## [155] 4621 4651 4681 4711 4741 4771 4801 4831 4861 4891 4921 4951 4981 5011
## [169] 5041 5071 5101 5131 5161 5191 5221 5251 5281 5311 5341 5371 5401 5431
## [183] 5461 5491 5521 5551 5581 5611 5641 5671 5701 5731 5761 5791 5821 5851
## [197] 5881 5911 5941 5971 6001 6031 6061 6091 6121 6151 6181 6211 6241 6271
## [211] 6301 6331 6361 6391 6421 6451 6481 6511 6541 6571 6601 6631 6661 6691
## [225] 6721 6751 6781 6811 6841 6871 6901 6931 6961 6991 7021 7051 7081 7111
## [239] 7141 7171 7201 7231 7261 7291 7321 7351 7381 7411 7441 7471 7501 7531
## [253] 7561 7591 7621 7651 7681 7711 7741 7771 7801 7831 7861 7891 7921 7951
## [267] 7981 8011 8041 8071 8101
```

```
colnames(newDset1)
```

```
## [1] "date"          "Temperature"   "Humidity"      "Light"
## [5] "CO2"           "HumidityRatio" "Occupancy"     "newLight"
```

```
## end of Pre-procesing

# defining variables
obsSeq <- newDset1[,"newLight"]


nrow(newDset1)
```

```
## [1] 271
```

```
M <- length(unique(obsSeq)) # No. of unique observations symbols
N <- length(unique(newDset1[,"Occupancy"])) # No. of unique Hidden States

Obsvals<- sort(unique(obsSeq)) # sorted unique observation symbols
qvals <- unique(newDset1[,"Occupancy"])  # unique Hidden States


# calling functions
lambda <- forwardbackward(obsSeq,maxiter=2000)
```

```
## [1] "True"
## [1] TRUE
## [1] TRUE
## [1] TRUE
## [1] 1745
```

```
# parameters of HMM just estimated ("learned")
a1 <- lambda[[2]]
b1 <- lambda[[4]]
g1 <- lambda[[5]]

#Viter <- Viterbi(obsSeq,a1,b1,g1)
#Viter[[2]] # back pointer

# get test data
OccTest[1:3,]
```

```
##                    date Temperature Humidity    Light      CO2
## 140 2015-02-02 14:19:00      23.700   26.272 585.2000 749.2000
## 141 2015-02-02 14:19:59      23.718   26.290 578.4000 760.4000
## 142 2015-02-02 14:21:00      23.730   26.230 572.6667 769.6667
##     HumidityRatio Occupancy
## 140   0.004764163         1
## 141   0.004772661         1
## 142   0.004765153         1
```

```
min(OccTest[,4])
```

```
## [1] 0
```

```
coln = which(colnames(OccTest)=="Light")
min(OccTest[,coln])
```

```
## [1] 0
```

```
max(OccTest[,coln])
```

```
## [1] 1697.25
```

```
newOccTest <- Discretize(Dset=OccTest,minv=0,maxv=1800,nObs=6,coln)
newTestset1 <- getDset(newOccTest,chngt)
```

```
##  [1]    1   31   61   91  121  151  181  211  241  271  301  331  361  391
## [15]  421  451  481  511  541  571  601  631  661  691  721  751  781  811
## [29]  841  871  901  931  961  991 1021 1051 1081 1111 1141 1171 1201 1231
## [43] 1261 1291 1321 1351 1381 1411 1441 1471 1501 1531 1561 1591 1621 1651
## [57] 1681 1711 1741 1771 1801 1831 1861 1891 1921 1951 1981 2011 2041 2071
## [71] 2101 2131 2161 2191 2221 2251 2281 2311 2341 2371 2401 2431 2461 2491
## [85] 2521 2551 2581 2611
```

```
###############
# defining variables
colnames(newOccTest)
```

```
## [1] "date"          "Temperature"   "Humidity"      "Light"        
## [5] "CO2"           "HumidityRatio" "Occupancy"     "newLight"
```

```
colnames(newTestset1)
```

```
## [1] "date"          "Temperature"   "Humidity"      "Light"        
## [5] "CO2"           "HumidityRatio" "Occupancy"     "newLight"
```

```
obsSeqTest <- newTestset1[,"newLight"]
obsSeqTest[1:10]
```

```
##  [1] 2 2 2 2 2 2 2 2 1 1
```

```
####################################
## Testing model with test dataset
####################################


Viter <- Viterbi(obsSeqTest,a1,b1,g1)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.9389538 0.7925942 0.6690485 0.5647604 0.4767283 0.4024182 0.3396912
## [2,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##           [,8]        [,9]        [,10]        [,11]        [,12]
## [1,] 0.2867417 0.008052632 0.0002261439 2.768787e-05 2.681258e-05
## [2,] 0.0000000 0.027807751 0.0269286661 2.607737e-02 2.525299e-02
##             [,13]        [,14]        [,15]        [,16]        [,17]
## [1,] 2.596495e-05 2.514412e-05 2.434924e-05 2.357949e-05 2.283408e-05
## [2,] 2.445467e-02 2.368158e-02 2.293294e-02 2.220796e-02 2.150590e-02
##             [,18]        [,19]        [,20]        [,21]        [,22]
## [1,] 2.211222e-05 2.141319e-05 2.073626e-05 2.008072e-05 1.944591e-05
## [2,] 2.082604e-02 2.016767e-02 1.953011e-02 1.891270e-02 1.831482e-02
##             [,23]        [,24]        [,25]        [,26]        [,27]
## [1,] 1.883117e-05 1.823586e-05 1.765937e-05 1.710111e-05 1.656049e-05
## [2,] 1.773583e-02 1.717515e-02 1.663219e-02 1.610640e-02 1.559723e-02
##             [,28]        [,29]        [,30]        [,31]        [,32]
## [1,] 1.603697e-05 1.552999e-05 1.503904e-05 1.456361e-05 1.410321e-05
## [2,] 1.510416e-02 1.462667e-02 1.416428e-02 1.371650e-02 1.328288e-02
##             [,33]        [,34]        [,35]        [,36]        [,37]
## [1,] 1.365737e-05 1.322562e-05 1.280752e-05 0.0003727981 0.0003146882
## [2,] 1.286297e-02 1.245634e-02 1.206255e-02 0.0000000000 0.0000000000
##             [,38]      [,39]        [,40]        [,41]        [,42]
## [1,] 0.0002656361 0.00022423 0.0001892781 0.0001597744 0.0001348695
## [2,] 0.0000000000 0.00000000 0.0000000000 0.0000000000 0.0000000000
##             [,43]        [,44]        [,45]        [,46]        [,47]
## [1,] 0.0001138467 9.610082e-05 2.575272e-06 6.901116e-08 1.938056e-09
## [2,] 0.0000000000 0.000000e+00 0.000000e+00 0.000000e+00 6.692590e-09
##             [,48]        [,49]       [,50]        [,51]        [,52]
## [1,] 5.193527e-11 4.383985e-11 3.70063e-11 3.123794e-11 2.636872e-11
## [2,] 0.000000e+00 0.000000e+00 0.00000e+00 0.000000e+00 0.000000e+00
##             [,53]        [,54]        [,55]      [,56]        [,57]
## [1,] 2.225849e-11 1.878894e-11 1.586021e-11 1.3388e-11 3.759782e-13
## [2,] 0.000000e+00 0.000000e+00 0.000000e+00 0.0000e+00 1.298347e-12
##             [,58]        [,59]        [,60]        [,61]        [,62]
## [1,] 1.055868e-14 1.292749e-15 1.251882e-15 1.212306e-15 1.173982e-15
## [2,] 1.257302e-12 1.217555e-12 1.179065e-12 1.141791e-12 1.105695e-12
##             [,63]        [,64]        [,65]        [,66]        [,67]
## [1,] 1.136869e-15 1.100929e-15 1.066125e-15 1.032422e-15 9.997840e-16
## [2,] 1.070741e-12 1.036892e-12 1.004113e-12 9.723698e-13 9.416303e-13
##             [,68]        [,69]        [,70]        [,71]        [,72]
## [1,] 9.681778e-16 9.375709e-16 9.079315e-16 8.792291e-16 8.514341e-16
## [2,] 9.118626e-13 8.830359e-13 8.551206e-13 8.280877e-13 8.019094e-13
##             [,73]        [,74]        [,75]        [,76]        [,77]
## [1,] 8.245178e-16 7.984524e-16 7.732109e-16 7.487675e-16 7.250967e-16
## [2,] 7.765587e-13 7.520094e-13 7.282362e-13 7.052145e-13 6.829206e-13
##             [,78]        [,79]        [,80]        [,81]        [,82]
## [1,] 7.021743e-16 6.799765e-16 6.584804e-16 6.376639e-16 6.175055e-16
## [2,] 6.613315e-13 6.404248e-13 6.201791e-13 6.005734e-13 5.815875e-13
##             [,83]        [,84]        [,85]        [,86]        [,87]
## [1,] 5.979843e-16 1.740598e-14 1.469282e-14 1.240257e-14 1.046932e-14
## [2,] 5.632019e-13 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##             [,88]
## [1,] 2.805528e-16
## [2,] 0.000000e+00
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    1    1    1    1    1    1    1    1     1     2     2     2
## [2,]    0    1    1    1    1    1    1    1    1     2     2     2     2
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     2     2     2     2     2     2     2     2     2     2     2
## [2,]     2     2     2     2     2     2     2     2     2     2     2
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]     2     2     2     2     2     2     2     2     2     2     2
## [2,]     2     2     2     2     2     2     2     2     2     2     2
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]     2     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
##      [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## [1,]     1     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
##      [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66] [,67] [,68]
## [1,]     1     2     2     2     2     2     2     2     2     2     2
## [2,]     2     2     2     2     2     2     2     2     2     2     2
##      [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79]
## [1,]     2     2     2     2     2     2     2     2     2     2     2
## [2,]     2     2     2     2     2     2     2     2     2     2     2
##      [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88]
## [1,]     2     2     2     2     2     1     1     1     1
## [2,]     2     2     2     2     1     1     1     1     1
```

```
p <- Viter[[2]]
estbkPath <- Viter[[1]]

EstPathFromt1Test <- unpack(estbkPath,p)


ColnHS <- which(colnames(newTestset1)=="Occupancy") # Column index of Hidden states

#Accuracy on test data
ACCTestSet <- AccCalc(EstPathFromt1Test,newTestset1,ColnHS)  # Accuracy on Test set


############################################################
##########Test results other metrics & Confusion Matrix
###########################################################
getConfusionMat <- function(N,PredPath,GrTrPath){
  PredDf <- data.frame(matrix(rep(0,(N*length(PredPath))),ncol=N))
  qvals <- sort(qvals)
  colnames(PredDf) <- qvals

  for (c in 1:N){
    PredDf[which(PredPath==qvals[c]),c] <- 1
  }

  cnamesPredDf<- c()
  rnamesPredDf <- c()
  for (cntcls in 1:N){
    cnamesPredDf[cntcls] <- str_remove(paste("Pred_",qvals[cntcls])," ")
  }
  colnames(PredDf) <- cnamesPredDf
  GrTrDf <- data.frame(matrix(rep(0,(N*length(GrTrPath))),ncol=N))
  colnames(GrTrDf) <- qvals

  for (c in 1:N){
    GrTrDf[which(GrTrPath==qvals[c]),c] <- 1
  }

  PredDf <- cbind(PredDf,PredPath)
  confM <- matrix(rep(0,N*N),nrow=N)
  cnamesConfM <- c()
  rnamesConfM <- c()
  for (cntcls in 1:N){
    cnamesConfM[cntcls] <- str_remove(paste("Pred_",qvals[cntcls])," ")
    rnamesConfM[cntcls] <- str_remove(paste("GrTr_",qvals[cntcls])," ")
  }
  colnames(confM) <- cnamesConfM
  rownames(confM) <- rnamesConfM
  rnum <- nrow(PredDf)

  for (tcnt in 1:rnum){
    for (cP in 1: N){
      for (cr in 1:N){
        if ((PredDf[tcnt,cP]==1) && (GrTrDf[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 correctly.
  # 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])
    }
    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)
########################################################################

ConfusionMatrix <- getConfusionMat(N,EstPathFromt1Test,newTestset1[,ColnHS])
Metrics <- ReportMetrics(ConfusionMatrix)
```

```
## [1] "Accuracy:  95.45 %"
## [1] "Precision for class 0 : 93.1034482758621 %"
## [1] "Precision for class 1 : 100 %"
## [1] "Recall for class 0 : 100 %"
## [1] "Recall for class 1 : 88.2352941176471 %"
```

```
ConfusionMatrix
```

```
##        Pred_0 Pred_1
## GrTr_0     54      0
## GrTr_1      4     30
```

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