6450 BayesianMethods Project (Model File)

TeamPAJ (P.Jordan, A.Jassal, J.Imgrund)

25June2018

Contents


```
#
#
#
```

Environment Preparation

```
# Remove any objects in the Environment
rm(list = ls())

# Closes all of R's graphics windows
graphics.off()

# Knitr global options
library(knitr)
opts_chunk$set(eval = TRUE, echo = TRUE, warning = FALSE,
               tidy = TRUE, results = "hold", cache = TRUE)

# Load Necessary Libraries
suppressPackageStartupMessages(library(dplyr))

# Set the overall seed for reproducibility
set.seed(6450)
```

Working Directory

```
# initialize the working directory for the final project
jim_dir = "/Users/jimgrund/Documents/GWU/Bayesian_Methods/Final-Project/"
akash_dir = "C:/Users/akash/Desktop/GWU/6450_Bayesian_YHuang/project/project"
patrick_dir = "/Users/pjordan/Documents/GWU/6450/FinalProject"

for (directory in c(akash_dir, jim_dir, patrick_dir)) {
    if (dir.exists(directory)) {
        setwd(directory)
        break
    }
}


source("DBDA2E-utilities.R")
```

```
## Loading required package: coda
## Linked to JAGS 4.3.0
## Loaded modules: basemod,bugs
# ============================================================================
##
## ************************************************************************
## Kruschke, J. K. (2015). Doing Bayesian Data Analysis, Second Edition:
## A Tutorial with R, JAGS, and Stan. Academic Press / Elsevier.
## ************************************************************************
```

Data Load & Tidy

Load Data

```
# Load the csv into a dataframe and return that dataframe Params: filename
# of the csv to load Returns: dataframe containing the data
LoadData = function(filename) {
    df = read.csv(filename)
    return(df)
}
```

Feature Importance

```
# Random Forest Feature Importance
RFFeatureImportance = function(data) {
    transdf <- subset(data, select = c(2:32))
    transdf <- transdf[complete.cases(transdf), ]

    # Fit a random forest model
    library(randomForest)
    rf_fit <- randomForest(transdf$diagnosis ~ ., data = transdf)
    rf_fi <- data.frame(importance(rf_fit))

    return(rf_fi)
}
```

Bin the Parameters

```
# Create equally distributed bins for the parameters we're interested in
# Params: dataframe of data
BinData = function(df) {
    #' # Radius_mean
    # create three bins using 0, 12.25, 14.75, 30 as the breakpoints.
    df$radius_bin <- cut(df$radius_mean, breaks = c(0, 12.25, 14.75, 30), labels = 1:3)

    #' # Area_mean
    # create three bins on area_mean using 0, 463, 680, 2600 as the breakpoints.
    df$area_bin <- cut(df$area_mean, breaks = c(0, 463, 680, 2600), labels = 1:3)

    #' # Compactness_mean
```

```
    # create three bins on compactness_mean using 0, 0.075, 0.117, 0.5 as the
    # breakpoints.
    df$compactness_bin <- cut(df$compactness_mean, breaks = c(0, 0.075, 0.117,
        0.5), labels = 1:3)

    #' # Smoothness_mean
    # create three bins on smoothness_mean using 0, 0.0894, 0.102, 0.17 as the
    # breakpoints.
    df$smoothness_bin <- cut(df$smoothness_mean, breaks = c(0, 0.0894, 0.102,
        0.17), labels = 1:3)

    #' # Concavity_mean
    # create three bins on concavity_mean using 0, 0.039, 0.106, 0.43 as the
    # breakpoints.
    df$concavity_bin <- cut(df$concavity_mean, breaks = c(0, 0.039, 0.106, 0.43),
        labels = 1:3)

    #' # Symmetry_mean
    # create three bins on Symmetry_mean using 0, 0.167, 0.19, 0.35 as the
    # breakpoints.
    df$symmetry_bin <- cut(df$symmetry_mean, breaks = c(0, 0.167, 0.19, 0.35),
        labels = 1:3)

    return(df)
}
```

Plot Stacked Histogram

```
# Plot a stacked histogram to show ratio of malignant to benign in each bin
# Params: dataframe, column to hist on
PlotHistogram = function(df, param) {
    # plot a histogram just to see the distribution of those bins
    bartable <- table(df$diagnosis_code, df[[param]])
    barplot(bartable, xlab = param, ylab = "Frequency", main = "Stacked Barchart of Frequency vs Diagnos
        col = c("Purple", "Gold"), legend = c("B", "M"), cex.lab = 1.5, cex.axis = 1.5,
        cex.main = 1.5, cex.sub = 1.5)


    # define filename to save pdf as
    filename <- paste(paste(param, "_hist"), ".png", sep = "")

    # save a copy of the plot
    dev.copy(png, filename)
    dev.off()
}
```

MCMC

Generate

```
# returns coda data
genMCMC = function( data , zName="z" , NName="N" , sName="s" , cName="c" ,
```

```
                    numSavedSteps=50000 , saveName=NULL , thinSteps=1 ,
                    runjagsMethod=runjagsMethodDefault ,
                    nChains=nChainsDefault ) {
require(rjags)
require(runjags)
#------------------------------------------------------------------------------
# THE DATA.
# N.B.: This function expects the data to be a data frame,
# with one component z being a vector of integer # successes,
# one component N being a vector of integer # attempts,
# one component s being a factor of subject identifiers,
# and one component c being a factor of category identifiers, with
# subjects nested in categories.
z = data[[zName]]
N = data[[NName]]
s = data[[sName]]
c = data[[cName]]
Nsubj = length(unique(s))
Ncat =  length(unique(c))
# Specify the data in a list, for later shipment to JAGS:
dataList = list(
  z = z ,
  N = N ,
  c = as.numeric(c) , # c in JAGS is numeric, in R is possibly factor
  Nsubj = Nsubj ,
  Ncat = Ncat
)
#------------------------------------------------------------------------------
# THE MODEL.
modelString = "
model {
for ( sIdx in 1:Nsubj ) {
z[sIdx] ~ dbin( theta[sIdx] , N[sIdx] )
theta[sIdx] ~ dbeta( omega[c[sIdx]]*(kappa[c[sIdx]]-2)+1 ,
(1-omega[c[sIdx]])*(kappa[c[sIdx]]-2)+1 )
}
for ( cIdx in 1:Ncat ) {
omega[cIdx] ~ dbeta( omega0*(kappa0-2)+1 ,
(1-omega0)*(kappa0-2)+1 )
kappa[cIdx] <- kappaMinusTwo[cIdx] + 2
kappaMinusTwo[cIdx] ~ dgamma( 0.01 , 0.01 ) # mean=1 , sd=10 (generic vague)
}
omega0 ~ dbeta( 1.0 , 1.0 )
#omega0 ~ dbeta( 1.025 , 1.075 ) # mode=0.25 , concentration=2.1
kappa0 <- kappaMinusTwo0 + 2
kappaMinusTwo0 ~ dgamma( 0.01 , 0.01 )  # mean=1 , sd=10 (generic vague)
#kappaMinusTwo0 ~ dgamma( 1.01005 , 0.01005012 )  # mode=1 , sd=100
#kappaMinusTwo0 ~ dgamma( 1.105125 , 0.1051249 )  # mode=1 , sd=10
#kappaMinusTwo0 ~ dgamma( 1.105125 , 0.01051249 )  # mode=10 , sd=100
}
" # close quote for modelString
writeLines( modelString , con="TEMPmodel.txt" )
#------------------------------------------------------------------------------
# INTIALIZE THE CHAINS.
```

```
# Initial values of MCMC chains based on data:
initsList = function() {
  thetaInit = rep(NA,Nsubj)
  for ( sIdx in 1:Nsubj ) { # for each subject
    resampledZ = rbinom(1, size=N[sIdx] , prob=z[sIdx]/N[sIdx] )
    thetaInit[sIdx] = resampledZ/N[sIdx]
  }
  thetaInit = 0.001+0.998*thetaInit # keep away from 0,1
  kappaInit = 100 # lazy, start high and let burn-in find better value
  return( list( theta=thetaInit ,
                omega=aggregate(thetaInit,by=list(c),FUN=mean)$x ,
                omega0=mean(thetaInit) ,
                kappaMinusTwo=rep(kappaInit-2,Ncat) ,
                kappaMinusTwo0=kappaInit-2 ) )
}
#------------------------------------------------------------------------------
# RUN THE CHAINS
parameters = c( "theta","omega","kappa","omega0","kappa0")
adaptSteps = 500             # Number of steps to adapt the samplers
burnInSteps = 500            # Number of steps to burn-in the chains

useRunjags = TRUE
if ( useRunjags ) {
  runJagsOut <- run.jags( method=runjagsMethod ,
                          model="TEMPmodel.txt" ,
                          monitor=parameters ,
                          data=dataList ,
                          inits=initsList ,
                          n.chains=nChains ,
                          adapt=adaptSteps ,
                          burnin=burnInSteps ,
                          sample=ceiling(numSavedSteps/nChains) ,
                          thin=thinSteps ,
                          summarise=FALSE ,
                          plots=FALSE )
  codaSamples = as.mcmc.list( runJagsOut )
} else {
  # Create, initialize, and adapt the model:
  jagsModel = jags.model( "TEMPmodel.txt" , data=dataList , inits=initsList ,
                          n.chains=nChains , n.adapt=adaptSteps )
  # Burn-in:
  cat( "Burning in the MCMC chain...\n" )
  update( jagsModel , n.iter=burnInSteps )
  # The saved MCMC chain:
  cat( "Sampling final MCMC chain...\n" )
  codaSamples = coda.samples( jagsModel , variable.names=parameters ,
                              n.iter=ceiling(numSavedSteps*thinSteps/nChains),
                              thin=thinSteps )
}

# resulting codaSamples object has these indices:
#   codaSamples[[ chainIdx ]][ stepIdx , paramIdx ]
if ( !is.null(saveName) ) {
  save( codaSamples , file=paste(saveName,"Mcmc.Rdata",sep="") )
```

```
  }
  return( codaSamples )
} # end function

#==============================================================================
```

Summary Statistics

```
# returns summary output of coda
smryMCMC = function(codaSamples, compVal = 0.5, rope = NULL, diffSVec = NULL,
    diffCVec = NULL, compValDiff = 0, ropeDiff = NULL, saveName = NULL) {
    mcmcMat = as.matrix(codaSamples, chains = TRUE)
    summaryInfo = NULL
    rowIdx = 0
    # omega:
    for (parName in grep("omega", colnames(mcmcMat), value = TRUE)) {
        summaryInfo = rbind(summaryInfo, summarizePost(mcmcMat[, parName], compVal = compVal,
            ROPE = rope))
        rowIdx = rowIdx + 1
        rownames(summaryInfo)[rowIdx] = parName
    }
    # kappa:
    for (parName in grep("kappa", colnames(mcmcMat), value = TRUE)) {
        summaryInfo = rbind(summaryInfo, summarizePost(mcmcMat[, parName], compVal = NULL,
            ROPE = NULL))
        rowIdx = rowIdx + 1
        rownames(summaryInfo)[rowIdx] = parName
    }
    # theta:
    for (parName in grep("theta", colnames(mcmcMat), value = TRUE)) {
        summaryInfo = rbind(summaryInfo, summarizePost(mcmcMat[, parName], compVal = compVal,
            ROPE = rope))
        rowIdx = rowIdx + 1
        rownames(summaryInfo)[rowIdx] = parName
    }
    # differences of theta's:
    if (!is.null(diffSVec)) {
        Nidx = length(diffSVec)
        for (t1Idx in 1:(Nidx - 1)) {
            for (t2Idx in (t1Idx + 1):Nidx) {
                parName1 = paste0("theta[", diffSVec[t1Idx], "]")
                parName2 = paste0("theta[", diffSVec[t2Idx], "]")
                summaryInfo = rbind(summaryInfo, summarizePost(mcmcMat[, parName1] -
                  mcmcMat[, parName2], compVal = compValDiff, ROPE = ropeDiff))
                rowIdx = rowIdx + 1
                rownames(summaryInfo)[rowIdx] = paste0(parName1, "-", parName2)
            }
        }
    }
    # differences of omega's:
    if (!is.null(diffCVec)) {
        Nidx = length(diffCVec)
        for (t1Idx in 1:(Nidx - 1)) {
```

```
            for (t2Idx in (t1Idx + 1):Nidx) {
                parName1 = paste0("omega[", diffCVec[t1Idx], "]")
                parName2 = paste0("omega[", diffCVec[t2Idx], "]")
                summaryInfo = rbind(summaryInfo, summarizePost(mcmcMat[, parName1] -
                  mcmcMat[, parName2], compVal = compValDiff, ROPE = ropeDiff))
                rowIdx = rowIdx + 1
                rownames(summaryInfo)[rowIdx] = paste0(parName1, "-", parName2)
            }
        }
    }
    # save:
    if (!is.null(saveName)) {
        write.csv(summaryInfo, file = paste(saveName, "SummaryInfo.csv", sep = ""))
    }
    show(summaryInfo)
    return(summaryInfo)
}

# ================================================================================
```

Plot Results

```
# plot the analysis of the coda data
plotMCMC = function(codaSamples, data, zName = "z", NName = "N", sName = "s",
    cName = "c", compVal = 0.5, rope = NULL, diffSList = NULL, diffCList = NULL,
    compValDiff = 0, ropeDiff = NULL, saveName = NULL, saveType = "jpg") {
    #-------------------------------------------------------------------------------
    # N.B.: This function expects the data to be a data frame, with one
    # component z being a vector of integer # successes, one component N being a
    # vector of integer # attempts, one component s being a factor of subject
    # identifiers, and one component c being a factor of category identifiers,
    # with subjects nested in categories.
    z = data[[zName]]
    N = data[[NName]]
    s = data[[sName]]
    c = data[[cName]]
    Nsubj = length(unique(s))
    Ncat = length(unique(c))
    # Now plot the posterior:
    mcmcMat = as.matrix(codaSamples, chains = TRUE)
    chainLength = NROW(mcmcMat)

    # kappa:
    parNames = sort(grep("kappa", colnames(mcmcMat), value = TRUE))
    nPanels = length(parNames)
    nCols = 4
    nRows = ceiling(nPanels/nCols)
    openGraph(width = 2.5 * nCols, height = 2 * nRows)
    par(mfcol = c(nRows, nCols))
    par(mar = c(3.5, 1, 3.5, 1), mgp = c(2, 0.7, 0))
    # xLim = range( mcmcMat[,parNames] )
    xLim = quantile(mcmcMat[, parNames], probs = c(0, 0.995))
    mainLab = c(levels(myData[[cName]]), "Overall")
```

```
mainIdx = 0
for (parName in parNames) {
    mainIdx = mainIdx + 1
    postInfo = plotPost(mcmcMat[, parName], compVal = compVal, ROPE = rope,
        xlab = bquote(.(parName)), cex.lab = 1.25, main = mainLab[mainIdx],
        cex.main = 1.5, xlim = xLim, border = "skyblue")
}
if (!is.null(saveName)) {
    saveGraph(file = paste(saveName, "Kappa", sep = ""), type = saveType)
}

# omega:
parNames = sort(grep("omega", colnames(mcmcMat), value = TRUE))
nPanels = length(parNames)
nCols = 4
nRows = ceiling(nPanels/nCols)
openGraph(width = 2.5 * nCols, height = 2 * nRows)
par(mfcol = c(nRows, nCols))
par(mar = c(3.5, 1, 3.5, 1), mgp = c(2, 0.7, 0))
# xLim = range( mcmcMat[,parNames] )
xLim = quantile(mcmcMat[, parNames], probs = c(0.001, 0.999))
mainLab = c(levels(myData[[cName]]), "Overall")
mainIdx = 0
for (parName in parNames) {
    mainIdx = mainIdx + 1
    postInfo = plotPost(mcmcMat[, parName], compVal = compVal, ROPE = rope,
        xlab = bquote(.(parName)), cex.lab = 1.25, main = mainLab[mainIdx],
        cex.main = 1.5, xlim = xLim, border = "skyblue")
}
if (!is.null(saveName)) {
    saveGraph(file = paste(saveName, "Omega", sep = ""), type = saveType)
}

# Plot individual omega's and differences:
if (!is.null(diffCList)) {
    for (compIdx in 1:length(diffCList)) {
        diffCVec = diffCList[[compIdx]]
        Nidx = length(diffCVec)
        temp = NULL
        for (i in 1:Nidx) {
            temp = c(temp, which(levels(myData[[cName]]) == diffCVec[i]))
        }
        diffCVec = temp
        openGraph(width = 2.5 * Nidx, height = 2 * Nidx)
        par(mfrow = c(Nidx, Nidx))
        xLim = range(c(compVal, rope, mcmcMat[, paste0("omega[", diffCVec,
            "]")]))
        for (t1Idx in 1:Nidx) {
            for (t2Idx in 1:Nidx) {
              parName1 = paste0("omega[", diffCVec[t1Idx], "]")
              parName2 = paste0("omega[", diffCVec[t2Idx], "]")
              if (t1Idx > t2Idx) {
                # plot.new() # empty plot, advance to next
                par(mar = c(3, 3, 3, 1), mgp = c(2, 0.7, 0), pty = "s")
```

```
                nToPlot = 700
                ptIdx = round(seq(1, chainLength, length = nToPlot))
                plot(mcmcMat[ptIdx, parName2], mcmcMat[ptIdx, parName1],
                  cex.main = 1.25, cex.lab = 1.25, xlab = levels(myData[[cName]])[diffCVec[t2Idx]],
                  ylab = levels(myData[[cName]])[diffCVec[t1Idx]], col = "skyblue")
                abline(0, 1, lty = "dotted")
              } else if (t1Idx == t2Idx) {
                par(mar = c(3, 1.5, 3, 1.5), mgp = c(2, 0.7, 0), pty = "m")
                postInfo = plotPost(mcmcMat[, parName1], compVal = compVal,
                  ROPE = rope, cex.main = 1.25, cex.lab = 1.25, xlab = bquote(.(parName1)),
                  main = levels(myData[[cName]])[diffCVec[t1Idx]], xlim = xLim)
              } else if (t1Idx < t2Idx) {
                par(mar = c(3, 1.5, 3, 1.5), mgp = c(2, 0.7, 0), pty = "m")
                postInfo = plotPost(mcmcMat[, parName1] - mcmcMat[, parName2],
                  compVal = compValDiff, ROPE = ropeDiff, cex.main = 1.25,
                  cex.lab = 1.25, xlab = bquote("Difference of " * omega *
                    "'s"), main = paste(cName, " ", levels(myData[[cName]])[diffCVec[t1Idx]],
                    "-", levels(myData[[cName]])[diffCVec[t2Idx]]))
              }
            }
        }
        if (!is.null(saveName)) {
            saveGraph(file = paste0(saveName, "OmegaDiff", compIdx), type = saveType)
        }
    }
}

# Plot individual theta's and differences:
if (!is.null(diffSList)) {
    for (compIdx in 1:length(diffSList)) {
        diffSVec = diffSList[[compIdx]]
        Nidx = length(diffSVec)
        temp = NULL
        for (i in 1:Nidx) {
            temp = c(temp, which(myData[[sName]] == diffSVec[i]))
        }
        diffSVec = temp
        openGraph(width = 2.5 * Nidx, height = 2 * Nidx)
        par(mfrow = c(Nidx, Nidx))
        xLim = range(c(compVal, rope, mcmcMat[, paste0("theta[", diffSVec,
            "]")], z[diffSVec]/N[diffSVec]))
        for (t1Idx in 1:Nidx) {
            for (t2Idx in 1:Nidx) {
              parName1 = paste0("theta[", diffSVec[t1Idx], "]")
              parName2 = paste0("theta[", diffSVec[t2Idx], "]")
              if (t1Idx > t2Idx) {
                # plot.new() # empty plot, advance to next
                par(mar = c(3, 3, 3, 1), mgp = c(2, 0.7, 0), pty = "s")
                nToPlot = 700
                ptIdx = round(seq(1, chainLength, length = nToPlot))
                plot(mcmcMat[ptIdx, parName2], mcmcMat[ptIdx, parName1],
                  cex.lab = 1.25, xlab = s[diffSVec[t2Idx]], ylab = s[diffSVec[t1Idx]],
                  col = "skyblue")
                abline(0, 1, lty = "dotted")
```

```
                } else if (t1Idx == t2Idx) {
                  par(mar = c(3, 1.5, 3, 1.5), mgp = c(2, 0.7, 0), pty = "m")
                  postInfo = plotPost(mcmcMat[, parName1], compVal = compVal,
                    ROPE = rope, cex.main = 1.25, cex.lab = 1.25, xlab = bquote(.(parName1)),
                    main = paste0(s[diffSVec[t1Idx]], " (", c[diffSVec[t1Idx]],
                      ")"), xlim = xLim)
                  points(z[diffSVec[t1Idx]]/N[diffSVec[t1Idx]], 0, pch = "+",
                    col = "red", cex = 3)
                  text(z[diffSVec[t1Idx]]/N[diffSVec[t1Idx]], 0, bquote(list(z ==
                    .(z[diffSVec[t1Idx]]), N == .(N[diffSVec[t1Idx]]))), adj = c((z[diffSVec[t1Idx]]/
                    xLim[1])/(xLim[2] - xLim[1]), -3.25), col = "red")
                } else if (t1Idx < t2Idx) {
                  par(mar = c(3, 1.5, 3, 1.5), mgp = c(2, 0.7, 0), pty = "m")
                  postInfo = plotPost(mcmcMat[, parName1] - mcmcMat[, parName2],
                    compVal = compValDiff, ROPE = ropeDiff, cex.main = 0.67,
                    cex.lab = 1.25, xlab = bquote("Difference of " * theta *
                      "'s"), main = paste(sName, " ", s[diffSVec[t1Idx]],
                      " (", c[diffSVec[t1Idx]], ")", "\n -", s[diffSVec[t2Idx]],
                      " (", c[diffSVec[t2Idx]], ")"))
                  points(z[diffSVec[t1Idx]]/N[diffSVec[t1Idx]] - z[diffSVec[t2Idx]]/N[diffSVec[t2Idx]]
                    0, pch = "+", col = "red", cex = 3)
                }
              }
            }
            if (!is.null(saveName)) {
                saveGraph(file = paste0(saveName, "ThetaDiff", compIdx), type = saveType)
            }
        }
    }
}

# =============================================================================
```