GAZM

Dr. Dalzell

November 17, 2017

```
# The following workspace is an outcome of the DataCreation Markdown file
#load("DataSpace.RData")

# We can remove some things that we don't need here so that the code runs
# faster.

#rm(BDay2011DF, BDay2012DF, AllDays11, AllDays12, BDay2011, BDay2012, BDmatch, Day_Diff, Race_match, Sc

#save.image("PreGAZM.RData")
```

Create the Input for GAZM

For this simulation, we have File 2011 and File 2012. File 2011 is presumed to be error free. We need to format in the form that we need for the data. Our response variable is always in File 1 in the standard vocabulary, so in this case File 2012 is File 1.mWe need to create this data. We begin by putting the data in the form that we are most likely to get it in.

```
# Select the columns from the WholeThing that we need. For GAZM, we will
# need less than we do for my model. The blocking variables are ONLY
# needed to determine the blocks, which I have already done.
#
# File2012 = WholeThing[, c("mathscal", "birthyear", "ethnic", "sex")]
# # Bind on the block indicator and seed indicator
# File2012 <- cbind(File2012, Block = blocks2, Seed = Seeds)
#
# File2011 = WholeThing[, c("mathscal2011", "birthyear2011", "ethnic2011", "sex2011")]
# # Bind on the block indicator and seed indicator
# File2011 <- cbind(File2011, Block = blocks1)
```

For the GAZM script, it makes more sense to bind on the model matrix. The model matrix only changes in the sense that the File 2 data changes each time.

- Determine the block indices

```
# blocks = NULL for( i in 1:(2*N)){ blocks[i] =
# category.block(B[i,],length(d),d) }

# blocks1 = blocks[1:N] blocks2 = blocks[(N+1):(2*N)]
```

- Convert to block notation

```
# Block_Ops = c(blocks1, blocks2)
# Index_to_Block = Legal_Index = sort(unique(Block_Ops))
# Block_Ops = as.numeric(as.factor(Block_Ops))
# indices1 <- blocks1
# indices2 <- blocks2
# blocks1 <- Block_Ops[1:N]
# blocks2 <- Block_Ops[(N + 1):(2 * N)]
```

- Which blocks are occupied with Type 1 Seeds?

```
# The type 1 seeds are affiliated with the 2011 data.
# Type1Blocks = blocks2[type1Seeds]
#
# length(Type1Blocks)

# So the blocks we need work with
# NonEmpty <- setdiff(1:K, Type1Blocks)
# G.K = length(NonEmpty)
#
# # Start by taking the block indicator and setting it equal to 0 if it is a
# # seed only block.
#
# Gblocks1 = blocks1
# Gblocks2 = blocks2
#
# Gblocks1 = ifelse(Gblocks1 %in% NonEmpty, Gblocks1, 0)
# Gblocks2 = ifelse(Gblocks2 %in% NonEmpty, Gblocks2, 0)
#
# # Convert to the Block Notation. We note that an empty block is assigned
# # to block 1 by default since we don't care. We would like ths to be 0, so
# # we subtract 1
#
# Block_Ops = c(Gblocks1, Gblocks2)
# Index_to_Block = sort(unique(Block_Ops))
# Block_Ops = as.numeric(as.factor(Block_Ops))
# Block_Ops = Block_Ops - 1
# Gblocks1 <- Block_Ops[1:N]
# Gblocks2 <- Block_Ops[(N + 1):(2 * N)]
```

These blocks we do not need to work for the rest of the GAZM process.

- Determine how many records are in each block. Remember that for GAZM we only need the number of records in each block which are NOT type 1 seeds

```
# # Storage: How many records in each block?
# n.RIB = matrix(NA, nrow = 2, ncol = G.K)
# # Storage: Which records in each block?
# RIB = vector("list", length = 2)
# RIB[[1]] = vector("list", length = G.K)
# RIB[[2]] = vector("list", length = G.K)
#
# for (k in 1:G.K) {
#     RIB[[1]][[k]] <- which(Gblocks2 == k)
#     RIB[[2]][[k]] <- which(Gblocks1 == k)
# }
#
# n.RIB[1, ] <- sapply(RIB[[1]], length)
#
# n.RIB[2, ] <- sapply(RIB[[2]], length)
```

- Create the framework for File 1 and File 2 using the model matrix

```
# EthNew = WholeThing$ethnic
# for (i in 1:N) {
```

```
#     if (EthNew[i] == "P") {
#         EthNew[i] <- "M"
#     }
# }
# Model1 = lm(mathscal ~ mathscal2011 + (birthyear > 1996) + EthNew + sex, data = WholeThing)
#
# EthNew11 = WholeThing$ethnic2011
# for (i in 1:N) {
#     if (EthNew11[i] == "P") {
#         EthNew11[i] <- "M"
#     }
# }
#
# ModelSecondary = lm(mathscal2011 ~ (birthyear2011 > 1996) + EthNew11 + sex,
#     data = WholeThing)
#
# File2011 = cbind(mathscal = WholeThing$mathscal2011, model.matrix(ModelSecondary)[,
#     -1], Block = Gblocks1, Seed = Seeds)
# File2012 = cbind(mathscal = WholeThing$mathscal, model.matrix(Model1)[, -c(1:2)],
#     Block = Gblocks2, Seed = Seeds)
# colnames(File2011) = colnames(File2012)
```

- Add on individual identifiers, blockrows and Imputation pointer

```
# lambda = 1:N
# Imputed = 0
# BlockRow = 0
# File2011 = cbind(File2011, lambda, Imputed, BlockRow)
# File2012 = cbind(File2012, lambda, Imputed, BlockRow)
# rm(lambda)
```

- Impute to balance the blocks

```
# FileHolder = rbind(File2012[-c(type1Seeds), ], File2011[-c(type1Seeds), ])
# FileHolder = FileHolder[order(FileHolder[, "Block"]), ]
#
# make_imputed_holder <- function(data, ToImpute) {
#     # Step 1: Store only one row from each block
#     ImpHolder = data[!duplicated(data[, "Block"]), ]
#     # Step 2: Set all record-specific row entries to 0
#     ImpHolder[, c(ToImpute)] = ImpHolder[, "BlockRow"] = ImpHolder[, "Seed"] = ImpHolder[,
#         "lambda"] = 0
#     # Step 3: Set the binary imputed indicator to 1
#     ImpHolder[, "Imputed"] = 1
#     # Step 4: Change the row names
#     rownames(ImpHolder) = 1:nrow(ImpHolder)
#     # Step 4: Transpose ImpHolder so that each record is a column
#     ImpHolder = as.data.frame(t(ImpHolder))
#     # Step 5: Convert to a list
#     ImpHolder = as.list(ImpHolder)
#     # Return the matrix
#     return(ImpHolder)
# }
#
# add_imputed_rows <- function(in_Bl, ImpHolder1, ImpHolder2, RIB, File1_data,
```

```
#     File2_data) {
#
#     to.impute = in_Bl[1, ] - in_Bl[2, ]
#
#     F1 = which(to.impute < 0)
#     F2 = which(to.impute > 0)
#
#     # Add in rows for each of the blocks that need it in File 1
#     for (k in F1) {
#         timeshere = abs(to.impute[k])
#         holder = matrix(rep(ImpHolder1[[k]], timeshere), nrow = timeshere, byrow = T)
#         colnames(holder) = colnames(File1_data)
#         preLength = nrow(File1_data)
#         File1_data = rbind(File1_data, holder)
#         postLength = nrow(File1_data)
#         RIB[[1]][[k]] = c(RIB[[1]][[k]], (preLength + 1):postLength)
#     }
#
#     # Add in rows for each of the blocks that need it in File 2
#     for (k in F2) {
#         timeshere = abs(to.impute[k])
#         holder = matrix(rep(ImpHolder2[[k]], timeshere), nrow = timeshere, byrow = T)
#         colnames(holder) = colnames(File2_data)
#         preLength = nrow(File2_data)
#         File2_data = rbind(File2_data, holder)
#         postLength = nrow(File2_data)
#         RIB[[2]][[k]] = c(RIB[[2]][[k]], (preLength + 1):postLength)
#     }
#
#     new.list <- list(File1_data = File1_data, File2_data = File2_data, RIB = RIB)
#
#     return(new.list)
# }
#
# Imp.Holder1 = make_imputed_holder(FileHolder, 1)
# Imp.Holder2 = make_imputed_holder(FileHolder, 1)
#
# # This code takes a very long time to run because there are so many
# # records and so many blocks, though the new code has shortened it a
# # little.
# Holder = add_imputed_rows(n.RIB, Imp.Holder1, Imp.Holder2, RIB, File2012, File2011)
```

```
# rm(FileHolder)
# # The response variables
# File2012 = Holder$File1_data


# Update the records in block after the imputations

# for (i in (N + 1):nrow(File2012)) {
#     which.block = File2012[i, "Block"]
#     RIB[[1]][[which.block]] = c(RIB[[1]][[which.block]], i)
#
#     which.block = File2011[i, "Block"]
```

```
#     RIB[[2]][[which.block]] = c(RIB[[2]][[which.block]], i)
# }

# n.RIB[1, ] <- sapply(RIB[[1]], length)
#
# n.RIB[2, ] <- sapply(RIB[[2]], length)
```

*Create an indicator for whether or not a particular block needs a permutation (this should be everything except the Type 1 Seeds with our current structuring.)

```
# Set a threshold for exact sampling
# threshold = 5
#
# PermSampling = ifelse(n.RIB[1, ] > 1, ifelse(n.RIB[1, ] < threshold, 1, 2), 0)
# SeedsOnly = which(n.RIB == 0)  # Blocks which contain only seeds.
# needs.C = which(PermSampling > 0)
```

- Permute the records from 2011. We don't know which records from 2012 they match with

```
# G.get_starting_permutation <- function(K, RIB, matches, File1_data, File2_data,
#     n1, n2, type1seeds) {
#
#     holder1 = rep(0, n1)
#     holder2 = rep(0, n2)
#     lambdaholder = rep(0, n2)
#
#     for (k in 1:G.K) {
#
#         # print(k)
#
#         # Step 1: Look at the records in the block
#         in.block = list(RIB[[1]][[k]], RIB[[2]][[k]])
#         max.length = max(length(in.block[[1]]), length(in.block[[2]]))
#
#         # If the length is not 0, proceed. This means that we have some elements
#         # in the block.
#         if (max.length > 0) {
#
#             # Fill in the information from file 1
#             places1 = 1:length(in.block[[1]])
#             holder1[in.block[[1]]] = places1
#             # Fill in the information from file 2
#             places2 = permute.places(1:length(in.block[[2]]))
#             holder2[in.block[[2]]] = places2
#
#             # Fill in the value for lambda2
#             places2 = as.numeric(as.factor(places2))
#             if (length(places2) > 1) {
#                 lambdaholder[in.block[[2]]] = c(in.block[[1]])[order(places1)][c(places2)]
#                 # lambda_F1[ order(BlockRow_F1) ][result]
#             } else {
#                 lambdaholder[in.block[[2]]] = in.block[[1]]
#             }
#         }
#
```

```
#
#         if (length(unique(lambdaholder[which(lambdaholder > 0)])) != length(lambdaholder[which(lambda
#             0)])) {
#             print(k)
#             break
#         }
#     }
#
#     File1_data[, "BlockRow"] = holder1
#     File2_data[, "BlockRow"] = holder2
#     File2_data[, "lambda"] = lambdaholder
#     File2_data[type1seeds, "lambda"] = File1_data[type1seeds, "lambda"]
#
#     newlist <- list(File1 = File1_data, File2 = File2_data)
#
#     return(newlist)
# }
#
# permute.places <- function(vec1) {
#     n = length(vec1)
#     vec1.ordering = sample(1:n, n, replace = FALSE)
#     vec2 = vec1[order(vec1.ordering)]
#     vec2
# }
#
# nrow.F2011 = nrow.F2012 = nrow(File2011)
#
#
# File2011[, "lambda"] = 1:nrow.F2011
# File2012[, "lambda"] = 1:nrow.F2012
# holder = G.get_starting_permutation(G.K, RIB, matches, File2012, File2011, nrow.F2012,
#     nrow.F2011, type1Seeds)
# File2011 = holder$File2
# File2012 = holder$File1
# rm(holder)
```

- Run the GAZM

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ## Run the GAZM MCMC Only ####
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ###

# where.coefs.primary = c(1:8)
# where.coefs.secondary = c(2:8)
# where.coefs = c(1:8)
# p = length(where.coefs.primary) + 1
# p2 = length(where.coefs.secondary) + 1
#
# col.Y1 = 1
# col.Y2 = 1
#
# nonseed.F2 = which(File2011[, "Seed"] == 0)
# n.nonseed.F2 = length(nonseed.F2)
#
# # Set the Seed
```

```
# MCMCSeed = 1253782554
#
# # Set the iterations
# its = 5000
#
# # Burnin
# burnin = 1000
#
# # Gap
# gap = 50
#
# reps = 20
#
# realmin = 1e-15
#
# namepart = "GAZM"
#
# thinning = 2
#
# printgap = 100

# Run the script GAZM_mcmc('no',MCMCSeed, its,burnin=0, thinning, gap,
# n.RIB, RIB, File2012, File2011,G.K,which.Y2=2, which.Y1=1, type1Seeds,
# where.coefs.primary, where.coefs.secondary, p, p2,
# n.type1,needs.C,reps,PermSampling,secondary.option='dependent', n1=N,
# n2=N,col.Y1, col.Y2,namepart)
```

Evaluating the β draws

One of the major points of this method is to determine the relationship between the math score from 2011 and the math score from 2012.

```
# # Function: Summarize the Posterior Draws
# posterior_summaries <- function(object) {
#     apply(object, 2, function(x) mean(x) + 1.96 * sd(x))
#     apply(object, 2, function(x) mean(x) - 1.96 * sd(x))
#     cat("Posterior CI Lower: ", apply(object, 2, function(x) mean(x) - 1.96 * sd(x)), "\n")
#     cat("Posterior Means: ", c(apply(object, 2, mean)), "\n")
#     cat("Posterior CI Upper: ", apply(object, 2, function(x) mean(x) + 1.96 * sd(x)), "\n")
#     cat("Posterior SE: ", apply(object, 2, sd), "\n ")
# }
#
# # Load in the Stored Results
# load(paste("GOut", namepart, ".RData", sep = ""))
#
# posterior_summaries(G.outlist$out.coefs)
```

It is also useful to look at traceplots of the data. It is worth noting that the function below is supposed to compare the outputs from three different MCMC chains. If we only want to look at one, object,comp.object and test.object all need to be the same, ie the object in which you are interested.

```
make_traceplotsALL <- function(object, truth, name, comp.object, option, test.object) {
```

```
    max.point = max(max(object), max(comp.object), max(test.object), truth)
    min.point = min(min(object), min(comp.object), min(test.object), truth)

    plot(object, type = "l", ylab = "Parameter", ylim = c(min.point, max.point),
        main = paste("Trace Plot for ", name, sep = ""))

    # abline(h=1,col='blue',lwd=3) #Prior
    abline(h = truth, col = "green", lwd = 2)  # Target

    mcmc.mean = mean(object)
    mcmc.sd = sd(object)

    type.mean = ifelse(abs(truth - mcmc.mean) > 0.05 | abs(truth - mean(test.object)) >
        0.05, 1, 2)

    if (option == "compare") {
        comp.object.mean = mean(comp.object)
        comp.object.sd = sd(comp.object)
        # lines(comp.object, type='l')
        abline(h = comp.object.mean + 1.96 * comp.object.sd, col = "darkmagenta",
            lty = 5, lwd = 2)
        abline(h = comp.object.mean - 1.96 * comp.object.sd, col = "darkmagenta",
            lty = 5, lwd = 2)
        abline(h = comp.object.mean, col = "darkmagenta", lwd = 2, lty = type.mean)

        test.object.mean = mean(test.object)
        test.object.sd = sd(test.object)
        # lines(comp.object, type='l')
        abline(h = test.object.mean + 1.96 * comp.object.sd, col = "darkgreen",
            lty = 5, lwd = 2)
        abline(h = test.object.mean - 1.96 * comp.object.sd, col = "darkgreen",
            lty = 5, lwd = 2)
        abline(h = test.object.mean, col = "darkgreen", lwd = 2, lty = type.mean)
    }

    if (option == "COUNTER") {
        comp.object.mean = sum(apply(comp.object, 1, function(x) ifelse(sum(x) ==
            0, 0, 1)))
        cat("There are ", comp.object.mean, " records in the wrong block in the starting data.")
    }

    abline(h = mcmc.mean, col = "blue", lwd = 3, lty = type.mean)  # Posterior Mean
    abline(h = mcmc.mean + 1.96 * mcmc.sd, col = "blue", lty = 5, lwd = 2)
    abline(h = mcmc.mean - 1.96 * mcmc.sd, col = "blue", lty = 5, lwd = 2)


}

# # The Intercept
# make_traceplotsALL(G.outlist$out.coefs[, 1], 99.53767, "Intercept", G.outlist$out.coefs[,
#     1], "both", G.outlist$out.coefs[, 1])
```

Now we need to examine the match rate, ie how well we are doing at matching the records successfully.

```
# HOLDERG = apply(G.outlist$out.lambda, 1, function(x) sum(x[1:N] == 1:N))
# # Overall Match Rate
# summary(HOLDERG)/N
# NonSeed Match Rate
#posterior_summaries(as.matrix((HOLDERG/N - p.type1) * N/n.non))
```