Predict 450 Marketing Analytics Section 55
Solo 2 Project R Process Details

Chapter 1: Solo 2 Assignment Data Preparation Notes

This is a note describing how to prepare your data for analysis. A good way to proceed is step by step, checking your results as you go.

For this assignment you will estimate Hierarchical Bayes Multinomial Logit (MNL) regression models. To estimate your models, you will use the function rhierMnlDP() in the R package "bayesm." To use rhierMnlDP() you need to put your data into the required form. Your preparation will be in two steps. You will code a matrix of categorical predictor variables representing STC's choice task attributes and levels, and you'll create a list data structure for input to rhierMnlDP().

To do what follows, you'll need to have the R package "dummies" installed. You'll also need to have the Solo 2 data sets available and ready to import into R, and also the R data efcode.RData. It includes some simple R functions that you'll use to create predictor variables for your regression models.

Step 1: Getting the stuff you'll need

Launch R or RStudio. Install the package "dummies" if you haven't already done so, and attach it to your session with the library() or require command.

```
> require(dummies)
```

Read into your R session the following files. How you'll do this depends on the version of R and whether you are using an IDE like RStudio.

stc-cbc-respondents-v3.RData – This is the respondent data; R file format. The variables in it are documented in the file **stc-v3-datamap.txt.**

```
>  load("stc-cbc-respondents-v3(1).RData")
```

stb-dc-task-cbc-v3.csv – This describes the choice task in attributes and levels; it's a tab-delimited csv file with a header record. The attribute levels are coded 0,1,2,3. They correspond to the descriptions of the attributes and levels provided in the assignment description document.

stc-extra-scenarios-v3.csv – Descriptions of the two additional choice scenarios that you'll analyze after estimating your MNL model; a comma-delimited csv file.

efCode.RData – A couple of R functions you'll use to code your attributes and levels as the predictor variables you'll use in your MNL model; an R file.

```
> load("efCode.RData")
```

Step 2 : C reating your predictor variables

This is a longish step. What you are going to do here is to create the predictor variables for

your MNL model that reflect the attribute levels of the choice task alternatives . When you're done you'll have what's often called the design matrix, or sometimes the “X matrix,” in regression modeling parlance. (The dependent variable is often generically referred to as “Y,” not surprisingly.)

When categorical predictor variables are used in regression models, they are coded in particular ways so that it's possible to estimate regression coefficients for them. Common coding methods include dummy coding, effects coding, and contrast coding. We're going to use effects coding for this exercise, but generally speaking with any approach to coding you end up with one fewer coefficient estimate for a categorical predictor than the predictor has categories. (ie) If there are k levels for a categorical variable, you would need (k-1) dummy variables.

Effects coding consists of coding the categories of a predictor variable into new variables where there is one less new variable than there are categories, and one category is treated as a sort of reference category. The coded categories have values 1, 0, or -1.

This kind of coding is best explained by example. if gender is your predictor variable, for example, which has two categories, m and f, you could code it as a single variable where female is coded as 1, and male as -1. If you have a three category “size” predictor variable, say, with levels small, medium , and large, you could code it as *two* new variables, where medium might be coded as a 1 and a 0, large as a 0 and a 1, and small as -1 and -1. With four categories of color, say red, yellow, green, blue, you might code:

yellow:1,0,0
green: 0,1,0
blue: 0,0,1
red: -1, -1, -1

If we used the above predictors in a regression model, we'd get one estimated regression coefficient for gender, two for size, and three for color. The categories coded with -1 will not have their own coefficient estimates as a result of fitting a model.

Some software will automatically generate appropriate coding for variables used in models. For this assignment we're going to do some R programming and create our own.

The STC DC task whose data you'll be fitting models to has one attribute with four levels of categories, which is brand. It has four attributes with three levels each: price, RAM, CPU, and screen size. For the purpose of this assignment we're going to treat price as *both* categorical and continuous, categorical for investigating the effects of price levels per se on stated preference (the price level “main effects” in ANOVA jargon), and continuous to understand how the effect of price on preference may depend on brand. That is, we want to be able to assess whether brand and price *interact* in their effects on preference.

You may have used interaction terms in predictive models before. Interaction terms are just another kind of predictor variable. They allow looking into whether predictor variables “moderate” the effects of other predictor variables. Interactions can involve more than two

predictors. To estimate interaction effects using DCE data, the task needs to have been designed so as to provide enough information to estimate them. STC's task was designed so that brand by price interaction effects can be estimated.

We're going to proceed by first developing the coding for the attributes per se, the attribute "main effects." Then we're going to add to these coded predictors some additional predictors that will allow us to estimate brand by price interaction terms. All of these coded predictors will comprise your design matrix, "X.matrix," which you'll use for estimating your MNL regression models.

You could use the information above and what's in the file **stb-dc-task-cbc-v3.csv** to code your X matrix by hand. To make life a little easier for you, I've provided some R code to make the coding easier for you in **efCode.RData**. Once you imported this into your R session you'll see you have two R functions, efcode.att.f() and efcode.attmat.f(). (Note: I use the parens '()' to signify something that's an R function.)

The function efcode.att.f() accepts as input a numeric vector, xvec, and produces an effects coded version of it. This function requires the 'dummies' package. It uses the dummy() function in it. It assumes that the lowest value of xvec represents the reference category, the one that's coded with -1's. You can see what this function does by providing it with some vector input, e.g. xvec=c(0,1,2,3), efcode.att.f(xvec).

Important Note: ***This and other R functions are provided for your use, but they are not guaranteed to always work correctly. It's up to you to be sure you understand what they do, and to test them to see if what they produce makes sense.***

The function efcode.attmat.f() is a wrapper of sorts for efcode.att.f(). You can provide it with a matrix describing your choice design (like a matrix of the attribute columns in **stb-dc-task-cbc-v3.csv**), and it will produce an effects coded version of it. As indicated above, it's up to you to make sure you understand what this function is supposed to do and whether it is working correctly for you.

Here's how to use effcode.attmat.f(). Let's assume you have read in the choice task design that it's in **stb -dc-task-cbc-v3.csv** and that it's in a data frame called **taskV3**. Select the attribute variables in this data frame into a matrix we'll call **task.mat**. Then, do (where ">" is the R command prompt):

```
> taskV3 <- read.csv("stc-dc-task-cbc -v3(1).csv", sep="\t")
>  task.mat <- as.matrix(taskV3[, c("screen", "RAM", "processor", "price", "brand")])
>X.mat=efcode.attmat.f(task.mat)
```

You'll note that task.mat has 108 rows, 3 rows for each of the 36 choice questions, or "sets." X.mat should also have 108 rows.

If things have worked correctly, X.mat should be an effects coded version of task.mat. Check the results to see if it's correct by comparing rows in X.mat to rows in task.mat. Does it look like the 1's, 0's, and -1's, are in the right places in the rows of X.mat? In X.mat, each attribute is represented by one less column than it has levels. For example, the screen attribute is represented by the first two coded columns, RAM by the next two coded columns, and so on.

There should be 11 columns in X.mat.

So far we have coded attribute levels in X.mat, but we don't yet have any columns in it that represent the interaction between brand and price. If we think of price as a continuous variable rather than as categorical here, since it has three values we could estimate linear and quadratic price effects in a regression model. But to keep things simple for this assignment, we're only going to consider brand by *linear* price interaction terms.

We're going to create a new price variable that is centered on its mean, and then we are going to multiply it by each of the columns in X.mat that represent the brands. There are three such columns. You'll copy these columns into a separate matrix, and then multiply each of them by a vector of the price levels. You'll then take the resulting three new columns and combine them with X.mat.

First, get the vector of prices from **taskV3** and center it on its mean:

```
>pricevec=taskV3$price-mean(taskV3$price)
```

Note that the R scale() function could be used to do this, scale(taskV3$price,scale=FALSE), but the above is more transparent. Note also that the above is taking the element-wise difference of two numeric vectors.

Next, we'll get the columns from **X.mat** that represent brand. They should be the last three columns (check to be sure of this):

```
>X.brands=X.mat[,9:11]
```

Next, we're going to multiply each column in X.brands by pricevec. This isn't matrix multiplication. It's taking the *inner product* of each column and pricevec:

```
>X.BrandByPrice = X.brands*pricevec
```

Check this result. Try it first by multiplying a single X.brands column and pricevec.

Now we'll combine X.mat and X.BrandsByPrice to get the X matrix we'll use for choice modeling:

```
>X.matrix=cbind(X.mat,X.BrandByPrice)
```

This combines the columns of X.mat and X.BrandByPrice. Check your result, of course.

One way to see if your X matrix coding isn't seriously flawed is to premultiply X.matrix by its transpose, and then then to look at the determinant of the result. The transpose of X is X with its rows and columns reversed. X premultiplied by it's transpose is a square matrix, it has the same number of rows as columns. Without getting too detailed here, the determinant of a square matrix is the product of its eigenvalues. In any case, if in R you do:

>det(t(X.matrix)%*%X.matrix)

You should get a positive number, maybe a very big one. If you don't, go back through the steps above and check your results for each one.

Note that the %*% above means matrix multiplication in R. See for example,

https://en.wikipedia.org/wiki/Matrix_multiplication

That's your "linear algebra moment" for today. Welcome to The Matrix.

So now you have your X, or design, matrix, X.matrix, for doing some regression modeling. Next, you need to get the responses that STC's survey participants provided to the 36 choice questions they answered. The are in the data frame **resp.data-v3** that's in the R data file **stc-cbc-respondents-v3.RData.** The variable names are DCM1_1 through DCM1_36. They are probably the 4th through the 39th variables in **resp.data-v3**.

Let's get these responses out into their own data frame:

>ydata=resp.data[,4:39]

Check to see if you have all 36 response variables:

>names(ydata)

Make sure you have no missing data:

>ydata=na.omit(ydata)

Now, convert ydata to matrix

> *ydata=as.matrix(ydata)*

For one of the assignment's objectives you'll need a "covariate" (a variable) that indicates whether a survey respondent has owned a STC product. There's a variable in the respondent data set called **vList3.** Recode this into a variable that is equal to 1 if a respondent has ever owned an STC product. Otherwise, make it equal to zero.

zowner <- 1 * (! is.na(resp.data.v3$vList3))

In what follows I'll call this variable **zowner**.

Step 3: creating the necessary data structure

Now that we have our X matrix("**X.matrix**"**)**, our choice responses (they're in **ydata**), and

zowner, our indicator variable for STC product ownership. The next thing to do is to make the data structure you'll use as input to the bayesm package function rhierMnlDP(), which is what you'll use to estimate your HB MNL models. See help(rhierMnlDP) in R after you have attached bayesm with the library() or require command.

rhierMnlDP(), like some other functions in bayesm, takes its data in the form of of a "list."

require(bayesm)

See help(list). Lists are a type of data structure that can be found in a number of programming environments and languages. The list that rhierMnlDP() wants includes a scalar (single number) indicating the number of alternatives in each choice set, a list that contains lists with the X.matrix and y (response) data in it, and optionally, covariates. (**zowner**, here.)

Let's make the list of data lists, first. We're going to call it **lgtdata** to be consistent with the documentation for rhierMnlDP(). **lgtdata** is a list of data for each respondent. Its length equals the number of respondents. (Lists have length.) The data for each respondent is a list with two elements, **X.matrix** and the respondent's choice responses, from their row in ydata.

So, **lgtdata** is a list of lists.

Here's how you can create **lgtdata:**

```
> lgtdata = NULL # a starter placeholder for your list
> for (I in 1:424) {
>         lgtdata[[i]]=list(y=ydata[i,],X=X.matrix)
>  }
```

Note that for the above code to work, you need to have the brackets correct. It's all case sensitive too, of course. That "X" is upper case X. "y" is lower case y.

Assuming that the code runs without problems, you can check to see whether **lgtdata** has the correct length:

```
>length(lgtdata)
```

The length should be the number of respondents you have.

Next, you can look at data in **lgtdata**. If you wanted to look at the data for the 3rd respondent, for example, you could do:

```
> lgtdata[[3]]
```

You should get 3rd respondent's choices, which will be 1's, 2's, and 3's, in a vector **y**. You should also get an **X** that's **X.matrix**.

When you run rhierMnlDP(), you'll provide it with **lgtdata** and some other information, like

zowner. We'll be putting those pieces together in our Sync sessions, in some additional notes, and in our discussions in the Solo Huddle.
##

Putting it all together:
Below is the R code of what we have done so far.

```
load("stc-cbc-respondents-v3(1).RData")
ls()
str(resp.data.v3)
taskV3 <- read.csv("stc-dc-task-cbc -v3(1).csv", sep="\t")
str(taskV3)
require(dummies)
load("efCode.RData")
ls()
str(efcode.att.f)
str(efcode.attmat.f)
str(resp.data.v3)
str(taskV3)
apply(resp.data.v3[4:39], 2, function(x){tabulate(na.omit(x))})
task.mat <- as.matrix(taskV3[, c("screen", "RAM", "processor", "price", "brand")])
dim(task.mat)
head(task.mat)
X.mat=efcode.attmat.f(task.mat)  # Here is where we do effects coding
dim(X.mat)
head(X.mat)
pricevec=taskV3$price-mean(taskV3$price)
head(pricevec)
str(pricevec)
X.brands=X.mat[,9:11]
dim(X.brands)
str(X.brands)
X.BrandByPrice = X.brands*pricevec
dim(X.BrandByPrice)
str(X.BrandByPrice)
X.matrix=cbind(X.mat,X.BrandByPrice)
dim(X.matrix)
str(X.matrix)
X2.matrix=X.matrix[,1:2]
dim(X2.matrix)
det(t(X.matrix) %*% X.matrix)
ydata=resp.data.v3[,4:39]
names(ydata)
str(ydata)
ydata=na.omit(ydata)
str(ydata)
ydata=as.matrix(ydata)
```

```
dim(ydata)
zowner <- 1 * ( ! is.na(resp.data.v3$vList3) )
lgtdata = NULL
for (i in 1:424) { lgtdata[[i]]=list( y=ydata[i,],X=X.matrix )}
length(lgtdata)
str(lgtdata)
#########################################################################
```

Chapter 2: Solo 2 Modeling – Fitting HB MNL Model

The following assumes you have prepared the data as described in Chapter 1. This means you should have created the data objects X.matrix, lgtdata, and zowner, and that you are running an R session into which you have made them available for use.

First, install the package "bayesm" from an R mirror if you haven't already. Once it is installed, attach it your session using using library(bayesm).

```
>require(bayesm)
```

If you type

```
>help(package="bayesm")
```

You should see what's in this package. The function in bayesm you're going to use is rhierMnlDP(), which uses MCMC to fit a hierarchical MNL model. Type help(rhierMnlDP) to see what it does, how it works, and what it outputs. It produces a list of results, so when you run it, you are going to assign what it outputs to a name. Note that the DP part of the name refers to "Dirichlet Process." This code uses A Dirichlet prior to add additional "fatness" as needed to the tails of the multivariate normal (MVN) distribution, which can be a little too thin in this kind of choice modeling application.

Next, you need to create inputs for rhierMnlDP(). These consist of data, and specifications telling rhierMnlDP() how it should run. We're first going to try running with some test data, and for a relatively small number of iterations.

Subset lgtdata to get the first 100 respondents:

```
>lgtdata100=lgtdata[1:100]            # note the single [ ] here, not double
```

You're going to specify just 5,000 iterations, and that every 5th sample is kept:

```
>mcmctest=list(R=5000,keep=5)
```

Create the "Data" list rhierMnlDP() expects:

```
>Data1=list(p=3,lgtdata=lgtdata100   # p is choice set size
```

OK, so no we're ready to do a test run. Let's call the results testresults:

>testrun1=rhierMnlDP(Data=Data1,Mcmc=mcmctest)

You're using most of rhierMnlDP()'s defaults for priors, etc.

This will take a little bit to run. You'll get some process feedback. (Or error messages.)

If things are starting to work, you'll see output in the R console window like:

```
> testrun1=rhierMnlDP(Data=Data1,Mcmc=mcmctest) Z
not specified
 Table of Y values pooled over all
units ypooled
    1 2 3
  997 1034 1569

  Starting MCMC Inference for Hierarchical
    Logit: Dirichlet Process Prior
    3 alternatives; 14 variables in X
    for 100 cross-sectional units

  Prior Parms:
   G0 ~ N(mubar,Sigma (x) Amu^-1)
    mubar = 0
    Sigma ~ IW(nu,nu*v*I) Amu
    ~ uniform[ 0.01 , 2 ]
    nu ~ uniform on log grid [ 14.01005 , 33.08554 ]
    v ~ uniform[ 0.1 , 4 ]

   alpha ~ (1-(alpha-alphamin)/(alphamax-alphamin))^power
    Istarmin = 1
   Istarmax = 10
alphamin = 0.10834
alphamax = 1.833977
power = 0.8


 MCMC Parms:
 s= 0.783  w=  0.1  R=  5000  keep=  5  maxuniq=  200 gridsize for
 lambda hyperparms=  20
 initializing Metropolis candidate densities for  100 units ...
   completed unit # 50
   completed unit #  100
 MCMC Iteration (est time to end - min)
   100 ( 2 )
  200  ( 2 )
   300  ( 2 )
   400  ( 2 )

   … and so on...
```

Once the algorithm has completed 5,000 iterations, your output in testrun1 will be a list with several

components:

```
> names(testrun1)
 [1] "betadraw"  "nmix"      "alphadraw" "Istardraw" "adraw"     "nudraw"
 [7] "vdraw"     "loglike"
```

"betadraw" is an array that has the draws (i.e. samples from marginal posterior distributions) for the regression coefficients:

```
> dim(testrun1$betadraw)
[1]  100   14 1000
```

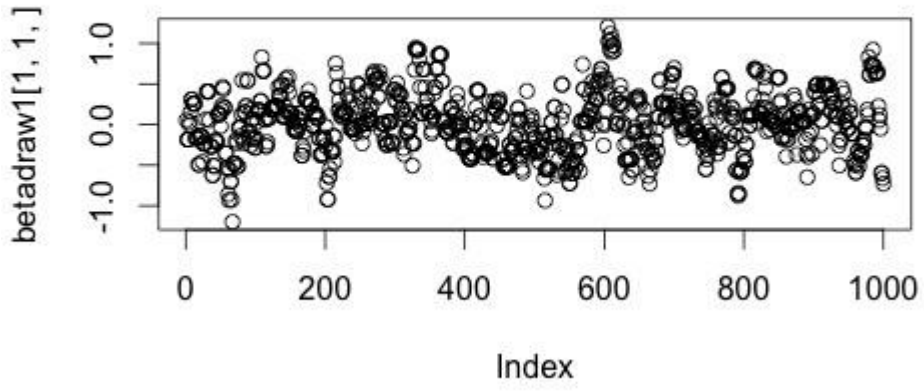
This array has 100 rows, the respondents, 14 columns corresponding to the columns in X.matrix, and 1000 "blocks," the third dimension of betadraw, which are the samples produced by the (thinned) iterations of the algorithm.

Let's get betadraw out of testrun1 and take a closer look at what's in it:

```
>  betadraw1=testrun1$betadraw  >
dim(betadraw1)
    [1]  100   14 1000

  > plot(1:length(betadraw1[1,1,]),betadraw1[1,1,])
```

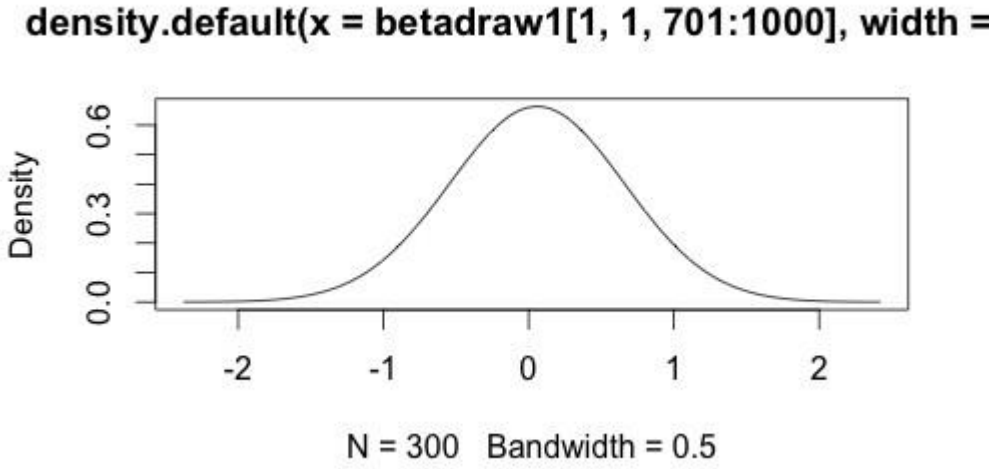
Here's what the chain in betadraw[1,1,] looks like using plot(betadraw1[1,1,]:



"Index" is the iteration number after "thinning."

These samples are what's used for making inferences about parameters. Taking a look at the last 300 sampled values:

```
>plot(density(betadraw1[1,1,701:1000],width=2))
```




```
 > summary(betadraw1[1,1,701:1000])
    Min.   1st Qu.    Median      Mean 3rd Qu.     Max.
-0.87680  -0.14560   0.05982   0.05453 0.23070  0.92140
```

We can summarize the samples of the coefficients. If we do the following, we get the overall 165 means of the coefficients, the means across respondents:

```
> apply(betadraw1[,,701:1000],c(2),mean)
 [1]  0.255145953  0.580166878 -0.085326056  0.476237786
110  1.011545742  1.170672099  0.259022260 -3.146228671
[9] -0.250393461  0.145450642 -0.003644152  0.192263456
    -0.045729188  0.015855950
```

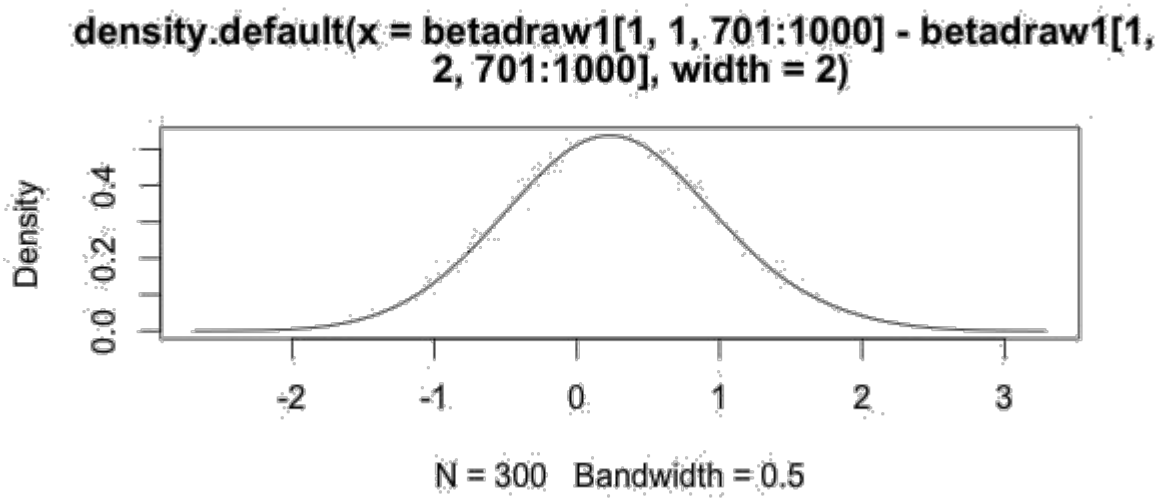
We can get a matrix of coefficient means by respondent with:

> apply(betadraw1[,,701:1000],c(1,2),mean)

this will produce a 100 rows x14 columns matrix: 100 respondents, 14 mean coefficients.

Since we have the chains, we can estimate functions of them that we want to evaluate. For example, we can calculate a distribution for the difference between respondent 1's 1st and 2nd coefficients (these are for the screen attribute):

```
> summary((betadraw1[1,1,701:1000]-betadraw1[1,2,701:1000]))
   Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
-1.1830 -0.1139  0.2320  0.2512  0.6287 1.7850
> plot(density(betadraw1[1,1,701:1000]betadraw1[1,2,701:1000],width=2))
```



##
Putting it all together:
Below is the R code of what we have done so far.

```
require(bayesm)
mcmctest=list(R=5000, keep=5)
Data1=list(p=3,lgtdata=lgtdata)
testrun1=rhierMnlDP(Data=Data1,Mcmc=mcmctest)
names(testrun1)
betadraw1=testrun1$betadraw
dim(betadraw1)
plot(1:length(betadraw1[1,1,]),betadraw1[1,1,])
plot(density(betadraw1[1,1,701:1000],width=2))
summary(betadraw1[1,1,701:1000])
betameansoverall <- apply(betadraw1[,,701:1000],c(2),mean)
betameansoverall
perc <- apply(betadraw1[,,701:1000],2,quantile,probs=c(0.05,0.10,0.25,0.5
,0.75,0.90,0.95))
perc
```

##

Chapter 3: Solo 2 Modeling – Fitting a HB MNL model with prior ownership as a covariate

The following assumes you have prepared the data and that you have worked through the previous steps, from Chapters 1 & 2.

You may be wondering what those other things are in the list of results that rhierMnlDP() returns. You probably saved your results in testrun1. What's in testrun1 is:

```
> names(testrun1)
[1] "betadraw"   "nmix"       "alphadraw" "Istardraw" "adraw"
```

```
[6] "nudraw"    "vdraw"     "loglike"
```

The help for rhierMnlDP describes what these things are. You'll see that "loglike" is the log-likelihood of the model at each iteration. You already know what "betadraw" is. "nmix" includes estimates of the the parameters of the MVN distribution on the betas. "Istardraw" summarizes the distribution of the number Dirichlet process (DP) components used per iteration. We've mentioned that the DP is used here to "fatten" MVN tails as needed. If you do:

```
>table(testrun1$Istardraw)
```

You'll see that the modal number is 3.

There are no results for how the betas might depend on whether a respondent had owned an STC product. That's because we didn't include **zowner**, the ownership indicator variable we created, as a covariate. Let's try doing that now.

To include zowner as a covariate, we need to create a new version of Data1. Let's call it Data2. Since rhierMnlDP() likes its z's centered, we're going to "demean" zowner and make the result a one column matrix (rhierMnlDP likes a matrix):

```
>zownertest=matrix(scale(zowner[1:100],scale=FALSE),ncol=1)
```

Note that we've included just the first 100 observations on zowner corresponding to the first 100 observations in lgtdata100.

```
>Data2=list(p=3,lgtdata=lgtdata100,Z=zownertest)
```

Now let's run the algorithm again, giving the output a new name so that we don't replace testrun1:

```
>testrun2=rhierMnlDP(Data=Data2,Mcmc=mcmctest)
```

```
> names(testrun2)
[1] "Deltadraw" "betadraw"  "nmix"      "alphadraw" "Istardraw"
[6] "adraw"     "nudraw"    "vdraw"     "loglike"
```

If things go ok, when the run is finished, testrun2 will now have in it

You can see that there is now a component called Deltadraw. Deltadraw is a matrix with number of rows=saved iterations (1000), and number of columns = number of regression predictors in your X.matrix (14):

```
> dim(testrun2$Deltadraw)
     1000   14
```

The columns are samples from the posterior distributions of the regression coefficients of the 14 betas on (mean-centered) zownertest. Here are the means of each column of Deltadraw:

```
> apply(testrun2$Deltadraw[701:1000,],2,mean)
 [1]  0.14255853 -0.08476901  0.04873744 -0.14364414 -0.43178944
 [6] -0.65383037 -0.10468255 0.21706991 0.47836681 -
0.11597557 [11] -0.74043095 0.31875305 0.16981103 -
```

```
0.10556410
```

Note that I've selected the last 300 draws from these chains to be consistent with what was done in the previous modeling note.

Here are selected "quantiles" of the last 300 draws:

```
>apply(testrun2$Deltadraw[701:1000,],2,quantile,probs=c(0.10,0.25,0.5
,0.75,0.90))

            [,1] [,2] [,3] [,4] [,5] 10% -0.08207724 -
  0.37552250 -0.14796340 -0.42673324 -0.76656529
 25%   0.01053868   -0.22790242   -0.06566431   -0.32960968   -
 0.64139878 50% 0.13700233 -0.07141960 0.04016023 -0.16397729
 -0.42413021
   75%    0.26596256    0.05356721    0.14980050    0.01758616  -0.26430172
   90%  0.41300909  0.17226737  0.29290200  0.14539719 -0.09768654
             [,6]          [,7]         [,8]       [,9]        [,10]
10% -0.9530506 -0.320185006 -0.38270880 0.2514029 -0.47396269
25% -0.8211855 -0.223487091 -0.06793703 0.3577536 -0.30089362
50% -0.6646355 -0.114702853 0.19091214 0.4643181 -0.10842736
   75%  -0.4958633   0.003250107    0.49778947   0.5950505   0.05024446
   90% -0.3253811  0.101493613  0.77800601  0.7253620 0.20365655
          [,11]       [,12]        [,13]        [,14]
   10% -1.1476781 0.03368964  -0.10720032 -0.37841654
   25% -0.9317426 0.16699358   0.03014244 -0.26747965
   50% -0.7533152 0.31074833   0.16853528 -0.13429931
   75% -0.5299044 0.46294954   0.30829174   0.02341187
   90% -0.3425735 0.62361721  0.44513902  0.22718009
```

You can think of the above as summarizing the posterior distributions of correlations between your choice model's regression coefficients and your zownertest variable. So, for example, the first regression coefficient, which corresponds to the coded screen size level 7 inches, and is [,1] above, is apparently positively related to previous STC product ownership. The coefficient for second level of processor speed on the other hand, [,5] above, is negatively related to previous STC ownership. So is the third level, [,6] above. You can identify other such relationships based on the above results. What do these relationships mean in terms of how preferences depend on STC product ownership experience?

Note the difference between deltadraw & betadraw. The betadraws from testrun2 provides the beta coefficients for the model with the covariate.

##

Putting it all together:
Below is the R code of what we have done so far.

```
zownertest=matrix(scale(zowner,scale=FALSE),ncol=1)
Data2=list(p=3,lgtdata=lgtdata,Z=zownertest)
testrun2=rhierMnlDP(Data=Data2,Mcmc=mcmctest)
dim(testrun2$deltadraw)
apply(testrun2$Deltadraw[701:1000,],2,mean)
```

```
apply(testrun2$Deltadraw[701:1000,],2,quantile,probs=c(0.05,0.10,0.25,0.5
,0.75,0.90,0.95))
betadraw2=testrun2$betadraw
dim(betadraw2)
#########################################################################
```

Chapter 4: Solo 2 Modeling – Make customer choice prediction using the individual respondent's model & goodness of fit & validation

Given that you have a design matrix X, and estimates of your choice model regression coefficients, your βi's, you can calculate predicted choice probabilities. In what follows we're going to step through a "quick and dirty" (Q&D) approach using the conditional logit model specification. It's Q&D because we're going to ignore the uncertainty in our βi's. Note that we don't have to do this; with a few more lines of R code we could take parameter uncertainty into account. But simple is good for this exercise and this assignment. But bear in mind that this approach comes with the risk thinking that the βi's may differ more than their uncertainties would indicate.

To start with let's assume that you have a design matrix of the kind we've previously called X.matrix. In what follows we're going to use the X.matrix we have, but you could instead use any design matrix that codes choice alternatives in the same manner. (For example, a design matrix coded to represent the alternatives Obee at STC is interested in.)

Using the "testrun2" results described in a previous note:

```
>betameans=apply(testrun2$betadraw[,,701:1000],c(1,2),mean)
```

betameans is a matrix with subjects in the rows (100, here) and means of regression coefficient draws in the columns (14).

```
>dim(betameans)
```

Next we're going to get the product of our X matrix and each subjects vector of mean β.'s Our X.matrix has 108 rows, and 14 columns. Each of the 36 choice sets had three alternatives. Now, if we calculate:

```
>xbeta=X.matrix%*%t(betameans)
```

This is matrix multiplication, the R symbol for which is "%*%." "t()" means matrix transpose. xbeta is an "X by β" matrix with 108 rows, one for each alternative in each choice set, and 100 columns, one for each of the 100 subjects.

We can reorganize xbeta so that subjects are in rows, with choice set alternatives across columns:

```
>xbeta2=matrix(xbeta,ncol=3,byrow=TRUE)
```

What matrix() is doing here is reading values column by column, and putting them into rows of a new matrix with three columns.

xbeta2 is 3600 by 3. We have 100 subjects who each responded to 36 choice sets. Note that based on this we know what the predicted choice is for the choice set in each row. It corresponds to the column with the largest value in it. You can use max.col() to find it for each row.

Next we can exponentiate the whole thing:

```
>expxbeta2=exp(xbeta2)
```

This is also a 3600 by 3 matrix. Since we have choice sets in rows, we can do the following to get predicted choice probabilities. What we want to do is to divide each row by its sum:

```
rsumvec=rowSums(expxbeta2)

pchoicemat=expxbeta2/rsumvec

custchoice <- max.col(pchoicemat)
```

phoicemat has 3600 rows, 108 for each of the 100 subjects. Its three columns are predicted choice probabilities. Note that this is like taking the *inner product* of each column of expxbeta2 and the reciprocal of rsumvec. (i.e., it's not matrix multiplication.) rowSums() is a function that sums across rows to get a vector. There is also a colSums() function. Custchoice provides the prediction of the customer choice.

```
head(custchoice)
str(custchoice)
```

You can use these results to calculate the likelihood, RLH (root likelihood, the geometric mean of the choice set likelihoods), ROC, AUC, MAE and MSE, and other measures that can be useful in assessing model fit.

```
ydatavec <- as.vector(t(ydata))
str(ydatavec)
table(custchoice,ydatavec)          ### provides confusion matrix

require("pROC")
roctest <- roc(ydatavec, custchoice, plot=TRUE)    ### ROC curve
auc(roctest)                    #### Area Under the Curve

logliketest <- testrun2$loglike    ## -2log(likelihood) test applies to nested models only
mean(logliketest)
```

Now you can use Custchoice to predict the choices for the 36 choice sets.

```
m <- matrix(custchoice, nrow =36,  ncol = 424)
m2 <- t(m)
apply(m2, 2, function(x){tabulate(na.omit(x))})
```

You can also apply the above steps to get Q&D predictions for held out responses. All you need to have is the appropriate X matrix.

Here's something to ponder. How would you generated predicted probabilities that reflect model parameter uncertainty, e.g. of the βi's?

##

Putting it all together:

Below is the R code of what we have done so far.

```
betameans <- apply(betadraw1[,,701:1000],c(1,2),mean)
str(betameans)
dim(betameans)
xbeta=X.matrix%*%t(betameans)
dim(xbeta)
xbeta2=matrix(xbeta,ncol=3,byrow=TRUE)
dim(xbeta2)
expxbeta2=exp(xbeta2)
rsumvec=rowSums(expxbeta2)
pchoicemat=expxbeta2/rsumvec
head(pchoicemat)
dim(pchoicemat)
custchoice <- max.col(pchoicemat)
str(custchoice)
head(custchoice)

ydatavec <- as.vector(t(ydata))
str(ydatavec)
table(custchoice,ydatavec)
require("pROC")
roctest <- roc(ydatavec, custchoice, plot=TRUE)
auc(roctest)
logliketest <- testrun2$loglike
mean(logliketest)

m <- matrix(custchoice, nrow =36,  byrow=F)
m2 <- t(m)
apply(m2, 2, function(x){tabulate(na.omit(x))})

##repeat this process for betadraw2##
```

##

Chapter 5: Solo 2 Prediction – Predicting extra scenarios, as well as the 36 choice sets, using betas from all the pooled respondents

Obee wants to predict the customer choices for the extra scenarios. You can use the logic described above, to make those predictions. Note that upto this point, we built models for each individual respondent. Also, the predictions, we did so far, were based on individual models. But if we want to predict new scenarios, which respondent's model would we use? There are 2 options – (1) Use individual respondent's models to predict the extra scenario & then use 'voting' kind of techniques to

make the final prediction, (2) We can pool the betas from all the respondents (ie betameansoverall) & then use that 1 model predict the extra scenario. If we want to use the pooled betas from all respondents to predict, then we can use the betameansoverall, which is the average value of the betas from all the respondents.

```
ex_scen <- read.csv("extra-scenarios.csv")
Xextra.matrix <- as.matrix(ex_scen[,c("V1","V2","V3","V4","V5","V6","V7","V8","V9",
"V10","V11","V12","V13","V14")])

betavec=matrix(betameansoverall,ncol=1,byrow=TRUE)
xextrabeta=Xextra.matrix%*%(betavec)
xbetaextra2=matrix(xextrabeta,ncol=3,byrow=TRUE)
dim(xbetaextra2)

expxbetaextra2=exp(xbetaextra2)
rsumvec=rowSums(expxbetaextra2)
pchoicemat=expxbetaextra2/rsumvec
pchoicemat
```

We can predict the original 36 choice sets using the pooled model. The code below, provides the probabilities as well as the frequencies for the 36 choice sets.

```
betavec=matrix(betameansoverall,ncol=1,byrow=TRUE)
xbeta=X.matrix%*%(betavec)
dim(xbeta)
xbeta2=matrix(xbeta,ncol=3,byrow=TRUE)
dim(xbeta2)
expxbeta2=exp(xbeta2)
rsumvec=rowSums(expxbeta2)
pchoicemat=expxbeta2/rsumvec
pchoicemat

pchoicemat2 <- round(pchoicemat*424,digits=0)
pchoicemat
```