Paper 1224-2017

A Moment-Matching Approach for Generating Synthetic Data in SAS®

Brittany Bogle, The University of North Carolina at Chapel Hill; Jared Erickson, SAS Institute Inc.

ABSTRACT

Disseminating data to potential collaborators can be essential in the development of models, algorithms, and innovative research opportunities. However, it is often time consuming to get approval to access sensitive data such as health data. An alternative to sharing the real data is to use synthetic data, which has similar properties to the original data but does not disclose sensitive information. The collaborators can use the synthetic data to make preliminary models or work out bugs in their code while waiting to get approval to access the original data. A data owner can also use the synthetic data to crowdsource solutions from the public through competitions like Kaggle and then test those solutions on the original data.

This paper implements a method that generates fully synthetic data in a way that matches the statistical moments of the true data up to a specified moment order as a SAS macro. Variables in the synthetic data set are of the same data type as the true data (for example, integer, binary, continuous). The implementation uses the linear programming solver within a column generation algorithm and the mixed integer linear programming solver from the OPTMODEL procedure in SAS/OR. The COFOR statement in PROC OPTMODEL automatically parallelizes a portion of the algorithm.

This paper demonstrates the method by using the Sashelp.Heart data set to generate fully synthetic data copies.

INTRODUCTION

Data owners face barriers in sharing sensitive data with potential collaborators. They might wish to share their data to promote interdisciplinary work or increase the utility of data that was expensive to collect, but struggle to gain permission to disseminate these data among external collaborators. Potential external collaborators might also be unexperienced in the data owner's domain and therefore might become frustrated with the time and effort required to complete the protocols and procedures necessary to acquire such data. These barriers often result in foregone innovative interdisciplinary opportunities and may hinder the advancement of research within application areas that harbor extensive protections on their data. However, collaborators who primarily aim to access these data in order to develop algorithms or modeling methods might actually be able to begin work without the real data, as long as an approximate, realistic data set can be used in lieu of the real data or during the development of analytical work while waiting for real data access. This alternative data is known as *synthetic data*. Researchers could use synthetic data to, for example, understand the format of the real data, develop an understanding of its statistical properties, build preliminary models, or tune algorithm parameters. Analytical procedures and code developed using the synthetic data can then be passed to the data owner if the collaborators are not permitted to access the original data. Even if the collaborators are eventually granted access, synthetic data allow the work to be conducted in parallel with tasks required to access the real data, allowing an efficient path to the final analysis.

This paper contains and describes an implementation of an algorithm to create synthetic data. The original algorithm can be found in *A Moment Matching Approach for Generating Synthetic Data* (Bogle 2016). The method generates data that have statistical marginal and mixed raw moments similar to the real data, up to the moment order specified by the user.

The data generation algorithm consists of three steps, which are implemented with the OPTMODEL procedure in SAS/OR. The preliminary step reads in the real data and computes its raw moments, which are used in the remaining steps. The linear programming (LP) step creates candidate synthetic observations by solving a series of linear programs. The integer programming (IP) step selects the desired number of synthetic observations from the candidates that most closely match the moments of the original data set.

USAGE

The macro to generate synthetic data is called %GENDATA and has parameters that can be used to control the algorithm by changing the size of the output, the time needed to run the algorithm, and the quality of the synthetic data.

Required Parameters

INPUTDATA=*inputdataset*

Specifies the name of the input data set that the synthetic data set emulates. While it is possible to customize the macro, two key assumptions are made about INPUTDATA for using the macro as written. The first assumption is that the original data has no missing values. The SAS DATA step can be used to remove rows with missing values. Note, however, that list-wise deletion (complete case analysis) could bias any analytical results. Using an appropriate imputation technique to impute missing data before using this macro is recommended. The second assumption is that the data are all numeric; categorical data can be represented as a binary or integer variable. For example, a two factor categorical variable, such as *Sex* represented as “Male” and “Female”, can be converted to “0” and “1” prior to using this macro. The Examples section demonstrates this.

METADATA=*metadataset*

Specifies the name of the user-created data set containing a column called “varName” with the names of the variables from INPUTDATA to be used and a column called “type” with the numeric type for each variable. The options are "c" for continuous, "i" for integer, and "b" for binary.

Optional Parameters

OUTPUTDATA=*SyntheticData*

Specifies the name of the synthetic data set created by the macro. The default value is *SyntheticData,* which is placed in the work folder.

MOMENTORDER=2

Specifies the order up to which moments are matched. The values are typically 2, 3, or 4. Larger values increase the size of the optimization problems, and therefore the computational time. The default value is 2.

NUMOBS=0

Specifies the desired number of observations to create for the synthetic data set. The default value of 0 generates a synthetic data set with the same number of observations as INPUTDATA.

MINNUMIPCANDS=0

Specifies the minimum number of observations to create during the LP step to pass to the IP step. The procedure usually produces more than this because they are selected in batches. The default value of 0 sets it to twice the value of NUMOBS. It must not be smaller than NUMOBS, and larger values produce better synthetic data. However, note that setting NUMOBS=0 and MINNUMIPCANDS≥*x* is a valid operation as long as *x* is larger than the number of observations in INPUTDATA.

LPBATCHSIZE=10

Specifies the number of random observations to create before assessing the reduced cost of the candidate observations in order to select one to add to the model during the LP step. The default value is 10.

LPGAP=1E-3

Specifies the objective value threshold that must be achieved to stop adding observations to the LP. Once this threshold is reached, the algorithm moves on to the IP step if there are enough candidate observations, or starts with a new LP if more candidate observations are needed. The default value is 1E-3.

NUMCOFORTHREADS=1

Specifies the number of threads to be used in the COFOR loop during the LP step to create candidate observations for the IP step. The default value is 1.

MILPMAXTIME=600

Specifies the maximum time (in seconds) allowed for the MILP solver to select the final observation set. The MILP solver commonly does not reach the optimal solution in a reasonable amount of time, so this time limit stops the solver and returns the best solution found. The default value is 600 (10 minutes).

RELOBJGAP=1E-4

Specifies the largest relative difference between the best feasible solution and best lower bound for the MILP solver to declare optimality. The default value is 1E-4, which is the same as the default value for the MILP solver.

ALPHA=0.95

Specifies the confidence level used to calculate confidence upper and lower bounds on the moments of the real data. These bounds define the range for each synthetic data moment that is considered a match to its corresponding moment in the real data. For example, ALPHA=0.95 sets the upper bound to be approximately 1.96 times the standard error above the mean and the lower bound to be approximately 1.96 times the standard error below the mean. ALPHA must be strictly greater than 0 and strictly less than 1, with smaller values causing the moments to be matched more closely. The default value is 0.95.

RANDSEED=0

Specifies the random number seed to use for producing candidate observations. A value of 0 uses a random number stream based on the system clock, and values greater than 0 use a reproducible stream. This option should only be used when NUMCOFORTHREADS=1. The default value is 0.

Macro Overview

An example call of the macro is:

```
%GENDATA(INPUTDATA       = OriginalData,
         METADATA        = Metadata,
         MOMENTORDER     = 4,
         NUMCOFORTHREADS = 3);
```

The macro is broken into three parts within the call to PROC OPTMODEL, each a macro itself, plus setting the random number seed. Only the relevant parameters are passed into the macro for each of these parts. Note that the macro for each of these parts is not intended to be run independently, as PROC OPTMODEL objects from previous steps are expected to be available. Here is the definition of the overall macro:

```
%macro GENDATA(INPUTDATA, METADATA, OUTPUTDATA=SyntheticData,
   MOMENTORDER=3, NUMOBS=0, MINNUMIPCANDS=0, LPBATCHSIZE=10, LPGAP=1E-3,
   NUMCOFORTHREADS=1, MILPMAXTIME=600, RELOBJGAP=1E-4, ALPHA=0.95,
   RANDSEED=0);
```

```
   proc optmodel printlevel=0;
      call streaminit(&RANDSEED);
      %PRELIMINARYSTEP(INPUTDATA=&INPUTDATA, METADATA=&METADATA,
         MOMENTORDER=&MOMENTORDER, ALPHA=&ALPHA);
      %LPSTEP(MOMENTORDER=&MOMENTORDER, NUMOBS=&NUMOBS,
         MINNUMIPCANDS=&MINNUMIPCANDS, LPBATCHSIZE=&LPBATCHSIZE,
         LPGAP=&LPGAP, NUMCOFORTHREADS=&NUMCOFORTHREADS);
      %IPSTEP(OUTPUTDATA=&OUTPUTDATA, MOMENTORDER=&MOMENTORDER,
         NUMOBS=&NUMOBS, MINNUMIPCANDS=&MINNUMIPCANDS,
         MILPMAXTIME=&MILPMAXTIME, RELOBJGAP=&RELOBJGAP);
   quit;
%mend GENDATA;
```

PRELIMINARY STEP

The preliminary step reads the original data into PROC OPTMODEL. It declares and computes sets and parameters to be used in the LP and IP steps. The most significant outputs from this step are the values of the statistical raw marginal and mixed moments of the real data, up to the specified order, and the desired lower and upper bounds for the moments of the synthetic data. A raw mixed moment is the mean value over all the observations of the monomials created by the variables. The acceptable ranges here are computed by using the standard error, but this can easily be customized.

For example, if there are two variables, the following list shows all monomials up to the third order:

$$1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3.$$

The raw moments of a population would be the expected values of these monomials. The raw moments of a data set are the mean values of the monomials over the observations.

Two SAS functions, not specific to PROC OPTMODEL, are used in this step. The STD function computes the standard deviation. The PROBIT function computes the ALPHA quantile from the standard normal distribution. PROBIT returns the one-sided value, but it is converted in the macro to a two-sided value (for example, ALPHA=0.95 uses PROBIT(0.975)=1.96).

Here is the code for this step:

```
%macro PRELIMINARYSTEP(INPUTDATA, METADATA, MOMENTORDER, ALPHA);
   /* Get number of variables to be used from metadata */
   %global dset nVars;
   %let dset=&METADATA;
   %let dsid = %sysfunc(open(&dset));
   %if &dsid %then
   %do;
      %let nVars=%sysfunc(attrn(&dsid,nobs));
      %let rc = %sysfunc(close(&dsid));
   %end;
   /* Begin PROC OPTMODEL code */
   set <num> OBS;
   set <string> VARS;
   num dataVal {OBS, VARS};
   str varType {VARS};
   read data &METADATA into VARS=[varName] varType=type;
   read data &INPUTDATA into OBS=[_n_] {v in VARS}
      <dataVal[_n_,v]=col(v)>;
   num nvar = &nVars;
   set VARS_IDX = 0..nvar;
   str varIdx2varName {VARS_IDX};
   num varName2varIdx {VARS};
   num iVar init 0;
```

```
   for {v in VARS} do;
      iVar = iVar + 1;
      varIdx2varName[iVar] = v;
      varName2varIdx[v] = iVar;
   end;
   num dataValByIdx {OBS, VARS_IDX};
   for {ob in OBS} dataValByIdx[ob,0] = 1;
   for {ob in OBS, v in VARS}
      dataValByIdx[ob,varName2varIdx[v]] = dataVal[ob,v];
   num d = &MOMENTORDER;
   num numMom = comb(nvar+d,d);
   set MOM_IDX_SET = 1..numMom;
   /* Set exponents for each variable for each moment index */
   num exponent {MOM_IDX_SET, VARS_IDX};
   set EXPONENTS = {
      i0 in 0..d %do j=1 %to &nvars; , i&j in 0..d %end;
      : i0 %do j=1 %to &nvars; + i&j %end; = d
   };
   num momIdx init 0;
   for {<e0 %do j=1 %to &nvars; , e&j %end; > in EXPONENTS} do;
      momIdx = momIdx + 1;
      %do j=0 %to &nvars; exponent[momIdx,&j] = e&j; %end;
   end;
   num computedMoments {ob in OBS, mi in MOM_IDX_SET} =
      prod {j in VARS_IDX: exponent[mi,j] > 0}
      dataValByIdx[ob,j]^exponent[mi,j];
   /* Compute standard error of each moment */
   num nObs = card(OBS);
   num mean {mi in MOM_IDX_SET} =
      (sum {ob in OBS} computedMoments[ob,mi]) / nObs;
   num stdErr {MOM_IDX_SET};
   num computedMomentsSlice {OBS};
   for {mi in MOM_IDX_SET} do;
      for {ob in OBS} computedMomentsSlice[ob] = computedMoments[ob,mi];
      stdErr[mi] = std(of computedMomentsSlice[*]) / sqrt(nObs);
   end;
   /* Set bounds to ALPHA confidence interval */
   num momLb {mi in MOM_IDX_SET} =
      mean[mi] - probit(0.5 + &ALPHA/2) * stdErr[mi];
   num momUb {mi in MOM_IDX_SET} =
      mean[mi] + probit(0.5 + &ALPHA/2) * stdErr[mi];
%mend PRELIMINARYSTEP;
```

LP STEP

The second step is the LP step, which takes the information from the preliminary step and constructs a linear programming model. For a review of linear and integer programming, see (Shrijver 1998). The model weights the observations with a nonnegative variable value *Weight*, and the weights must sum to 1, as enforced by the constraint *WeightEQ1*. Because this step's goal is to quickly generate many candidate observations and avoid the computational burden introduced by a large integer program, observations are permitted to be unequally weighted in this step. To avoid duplicating the algebraic expression for weighted moments, an implicit variable, *WeightedMoment*, is used. Constraints *Upper* and *Lower* bound the weighted moment values by the bounds computed in the preliminary step. A nonnegative slack variable, *Eta*, represents the violation of either the upper or lower bound of the constraint (which corresponds to a moment). The objective is to minimize the sum of these violations.

After the above declarations, the model has been specified. Now, synthetic observations that populate the variable *Weight* need to be created. The process adds synthetic observations to the model until the solution has an objective value of LPGAP, which indicates that there is a way to weight the observations such that there is no violation to the desired bounds on the moments. This process is repeated until the total number of nonzero weighted observations is greater than or equal to the desired number of candidate observations for the IP step, given by the macro parameter MINNUMIPCANDS. If the parameter NUMCOFORTHREADS is used, PROC OPTMODEL can solve multiple LPs concurrently. The variables and constraints are automatically replicated across iterations, so there is no additional index needed for the iteration. The additional index is needed to store the solution for use outside the COFOR loop. Each iteration of the COFOR loop produces at least MINNUMIPCANDS/NUMCOFORTHREADS candidate observations, which guarantees at least MINNUMIPCANDS candidate observations in total. PROC OPTMODEL does not run the other statements in the loop concurrently, only the SOLVE statement. Since the LPs tend to solve quickly, the performance difference for more threads are most noticeable the LPs get large.

Candidate observations are created by using a uniform random variable between the least and greatest values for that variable in the original data. If the variable is to be integer or binary, it is rounded. The moment values for that observation are then computed. This is done once for the first observation to go in the LP. The LP is then iteratively solved, adding a new observation at each iteration, until the objective is less than LPGAP. After the first observation, there are LPBATCHSIZE candidate observations created in each iteration. Then the one with the best reduced cost is selected and added to the model, while the others are discarded. For more information on column generation, see (Desaulniers 2005). If all candidates have a nonnegative reduced cost, they are all discarded and a new batch is created. Once this process is completed, a set of at least MINNUMIPCANDS candidate synthetic observations are available for the IP step.

LP step code:

```
%macro LPSTEP(MOMENTORDER, NUMOBS, MINNUMIPCANDS, LPBATCHSIZE, LPGAP,
   NUMCOFORTHREADS);
   num minVal {v in VARS_IDX} = if (v=0) then 1
      else min {ob in OBS} dataValByIdx[ob,v];
   num maxVal {v in VARS_IDX} = if (v=0) then 1
      else max {ob in OBS} dataValByIdx[ob,v];
   str varIdxType {v in VARS_IDX} = if (v=0) then '1'
      else varType[varIdx2varName[v]];
   num nThreads = &NUMCOFORTHREADS;
   set THREADSET = 1..nThreads;
   num LPBATCHSIZE = &LPBATCHSIZE;
   set CANDOBS = 1..LPBATCHSIZE;
   num lpCandObVal {CANDOBS, VARS_IDX};
   num lpCandObMoms {CANDOBS, MOM_IDX_SET};
   num numLpObs init 0;
   set LPOBS = 1..numLpObs;
   num lpObVal {LPOBS, VARS_IDX};
   num lpObMoms {LPOBS, MOM_IDX_SET};
   var Weight {LPOBS} >= 0;
   var Eta {MOM_IDX_SET} >= 0;
   minimize Obj = sum {mi in MOM_IDX_SET} Eta[mi];
   impvar WeightedMoment {mi in MOM_IDX_SET} =
      sum {ob in LPOBS} lpObMoms[ob,mi]*Weight[ob];
   con Upper {mi in MOM_IDX_SET}:
      WeightedMoment[mi] - Eta[mi] <= momUb[mi];
   con Lower {mi in MOM_IDX_SET}:
      WeightedMoment[mi] + Eta[mi] >= momLb[mi];
   con WeightEQ1:
      sum {ob in LPOBS} Weight[ob] = 1;
   num minNumIpCands init &MINNUMIPCANDS;
```

```
if (minNumIpCands = 0) then do;
   minNumIpCands = if (&NUMOBS > 0) then 2*&NUMOBS else 2*nObs;
end;
num minNumIpCandsPerThread = ceil(minNumIpCands / nThreads);
num numIpCands {THREADSET} init 0;
set IPCANDS {t in THREADSET} = 1..numIpCands[t];
num ipObVal {t in THREADSET, IPCANDS[t], VARS_IDX};
num ipObMoms {t in THREADSET, IPCANDS[t], MOM_IDX_SET};
num rc {CANDOBS};
num bestRc;
num bestRcIdx;
set INTBINSET = {v in VARS_IDX: varIdxType[v] in {'i','b'}};
option nonotes;
cofor {t in THREADSET} do;
   do while (numIpCands[t] < minNumIpCandsPerThread);
      numLpObs = 1;
      for {v in VARS_IDX}
         lpObVal[1,v] = rand('UNIFORM', minVal[v], maxVal[v]);
      for {v in INTBINSET} lpObVal[1,v] = round(lpObVal[1,v]);
      for {mi in MOM_IDX_SET}
         lpObMoms[1,mi] = prod {j in VARS_IDX: exponent[mi,j] > 0}
         lpObVal[1,j]^exponent[mi,j];
      solve with LP / printlevel2=0;
      do while (Obj >= &LPGAP);
            /* Create LPBATCHSIZE candidate observations
               and find best reduced cost */
         for {candOb in CANDOBS} do;
            for {v in VARS_IDX} lpCandObVal[candOb,v] =
               rand('UNIFORM', minVal[v], maxVal[v]);
            for {v in INTBINSET} lpCandObVal[candOb,v] =
               round(lpCandObVal[candOb,v]);
            for {mi in MOM_IDX_SET} lpCandObMoms[candOb,mi] =
               prod {j in VARS_IDX: exponent[mi,j] > 0}
               lpCandObVal[candOb,j]^exponent[mi,j];
            rc[candOb] = -WeightEQ1.dual - sum {mi in MOM_IDX_SET}
               lpCandObMoms[candOb,mi]*(Upper[mi].dual+Lower[mi].dual);
         end;
         bestRc = min {candOb in CANDOBS} rc[candOb];
         /* Add a good observation to the model */
         if (bestRc < 0) then do;
            for {candOb in CANDOBS: rc[candOb] = bestRc}
               bestRcIdx = candOb;
            numLpObs = numLpObs + 1;
            for {v in VARS_IDX}
               lpObVal[numLpObs,v] = lpCandObVal[bestRcIdx,v];
            for {mi in MOM_IDX_SET}
               lpObMoms[numLpObs,mi] = lpCandObMoms[bestRcIdx,mi];
            solve with LP / printlevel2=0;
         end;
      end;
      /* Observations with positive weight are saved for IP step */
      for {ob in LPOBS: Weight[ob] > 1E-6} do;
         numIpCands[t] = numIpCands[t] + 1;
         for {v in VARS_IDX}
            ipObVal[t,numIpCands[t],v] = lpObVal[ob,v];
         for {mi in MOM_IDX_SET}
            ipObMoms[t,numIpCands[t],mi] = lpObMoms[ob,mi];
```

```
            end;
         end;
      end;
      option notes;
      /* Move all IP candidates to one set, IPCANDS[1] */
      num ipCandIdx;
      num oldIpCandIdx;
      for {t in THREADSET: t > 1} do;
         oldIpCandIdx = numIpCands[1];
         numIpCands[1] = numIpCands[1] + numIpCands[t];
         ipCandIdx = 0;
         do while (ipCandIdx < numIpCands[t]);
            ipCandIdx = ipCandIdx + 1;
            for {v in VARS_IDX} ipObVal[1,oldIpCandIdx+ipCandIdx,v] =
               ipObVal[t,ipCandIdx,v];
            for {mi in MOM_IDX_SET} ipObMoms[1,oldIpCandIdx+ipCandIdx,mi] =
               ipObMoms[t,ipCandIdx,mi];
         end;
      end;
      put 'Number of IP step candidate observations: ' numIpCands[1];
      drop Upper Lower WeightEQ1;
   %mend LPSTEP;
```

IP STEP

The final step is the IP step. The candidate observations resulting from the LP step are used in an integer program that selects the desired number of observations, NUMOBS, that best fit the desired moment ranges. Unlike the LP model, which minimizes the sum of violations, in the IP step the largest scaled violation is minimized. To this end, *Eta* is scaled and replaced by a new variable, *ScaledEta*, which is equivalent to *Eta* divided by the acceptable moment range. The *NumAssigned* constraint selects exactly NUMOBS observations from the candidate observations into OUTPUTDATA, and all the selected observations are weighted equally in the expression for the moment values.

Because integer programs are computationally more difficult to solve than linear programs, it could take too long to solve to optimality. As long as the objective is relatively small when the time limit is reached, the synthetic data set should behave similarly to the original data set.

IP step code:

```
   %macro IPSTEP(OUTPUTDATA, MOMENTORDER, NUMOBS, MINNUMIPCANDS, MILPMAXTIME,
      RELOBJGAP);
      num numSynthObs init &NUMOBS;
      if (numSynthObs = 0) then numSynthObs = nObs;
      num momRange {mi in MOM_IDX_SET} = momUb[mi] - momLb[mi];
      var Assigned {IPCANDS[1]} binary;
      var ScaledEta {MOM_IDX_SET} >= 0;
      var MaxError >= 0;
      minimize IpObj = MaxError;
      con MaxCon {mi in MOM_IDX_SET}:
         MaxError >= ScaledEta[mi];
      con UpperIP {mi in MOM_IDX_SET}:
         (1/numSynthObs) *
         sum {ob in IPCANDS[1]} ipObMoms[1,ob,mi]*Assigned[ob] -
         momRange[mi]*ScaledEta[mi] <= momUb[mi];
      con LowerIP {mi in MOM_IDX_SET}:
         (1/numSynthObs) *
         sum {ob in IPCANDS[1]} ipObMoms[1,ob,mi]*Assigned[ob] +
         momRange[mi]*ScaledEta[mi] >= momLb[mi];
```

```
   con NumAssigned:
      sum {ob in IPCANDS[1]} Assigned[ob] = numSynthObs;
   /* Set an initial solution, then solve */
   for {i in 1..numSynthObs} Assigned[i] = 1;
   for {mi in MOM_IDX_SET} ScaledEta[mi] = if momRange[mi] <= 0 then 0
      else max(((1/numSynthObs) *
      sum {ob in IPCANDS[1]} ipObMoms[1,ob,mi]*Assigned[ob]
      - momUb[mi]) / momRange[mi], momLb[mi] - (1/numSynthObs) *
      sum {ob in IPCANDS[1]} ipObMoms[1,ob,mi]*Assigned[ob]) /
      momRange[mi], 0);
   MaxError = max {mi in MOM_IDX_SET: momRange[mi] > 0} ScaledEta[mi];
   solve with MILP / maxtime=&MILPMAXTIME relobjgap=&RELOBJGAP
      heuristics=3 primalin;
   /* Save selected observations to data set */
   set FINALOBS = 1..numSynthObs;
   num finalObsVal {FINALOBS,VARS};
   num obIdx init 0;
   for {ob in IPCANDS[1]: Assigned[ob] > 0.5} do;
       obIdx = obIdx + 1;
       for {v in VARS}
          finalObsVal[obIdx,v] = ipObVal[1,ob,varName2varIdx[v]];
   end;
   create data &OUTPUTDATA(drop=tmpvar) from [tmpvar]=FINALOBS
      {j in VARS} <col(j)=finalObsVal[tmpvar,j]>;
%mend IPSTEP;
```

EXAMPLES

This example walks through the process of using the macro on the *Sashelp.Heart* data set. The *Sashelp.Heart* data set has some categorical variables and dependent variables, as well as variables with many missing values. To simply this example, only *Status, Sex, AgeAtStart, Height, Weight, Diastolic, Systolic, Smoking,* and *Cholesterol* are considered. Two of these variables, *Status* and *Sex*, are categorical variables that must be converted into numeric values (e.g. 0 and 1) before the macro can be executed. Thus, surrogate numeric variables, *Dead* and *Male*, respectively, are created. The user can convert them back to the original form if desired after the macro execution. For simplicity, in this example rows with missing values are ignored. The other variables take integer values, except that Height has some decimals, so *Metadata* is set accordingly. The DATA step creates *OriginalData* from *Sashelp.Heart*, and *Metadata* indicates the desired numeric type for each variable:

```
data OriginalData;
   set Sashelp.Heart(keep = Status Sex AgeAtStart Height Weight Diastolic
      Systolic Smoking Cholesterol);
   label AgeAtStart =;
   if cmiss(of _all_) then delete;
   if (Status = "Dead") then Dead = 1;
   else if (Status = "Alive") then Dead = 0;
   drop Status;
   if (Sex = "Male") then Male = 1;
   else if (Sex = "Female") then Male = 0;
   drop Sex;
run;

data Metadata;
   length varName $ 11;
   input varName $ type $;
   datalines;
AgeAtStart i
```

```
Height c
Weight i
Diastolic i
Systolic i
Smoking i
Cholesterol i
Dead b
Male b
;
```

SMALL EXAMPLE WITH BASIC VERIFICATION

This example generates a synthetic data set with 100 observations which matches marginal and mixed moments up to the second order, use three threads, allows one minute for the IP step, and uses the default values for the other parameters. Using these specifications, you can call the macro:

```
%GENDATA(INPUTDATA=OriginalData, METADATA=Metadata, NUMOBS=100,
   MOMENTORDER=2, MILPMAXTIME=60, RANDSEED=100);
```

This macro call produces a data set called *Work.SyntheticData*. To convert the binary variables back to the original format of categorical variables and to round *Height* to the nearest hundredth, you need to use another DATA step:

```
data FinalSyntheticData;
   set SyntheticData;
   label AgeAtStart = 'Age at Start';
   length Status $5 Sex $6;
   if (Dead = 1) then Status = "Dead";
   else Status = "Alive";
   drop Dead;
   if (Male = 1) then Sex = "Male";
   else Sex = "Female";
   drop Male;
   Height = round(100*Height) / 100;
run;
```

The data set *FinalSyntheticData* now has 100 observations with similar properties to the original data set, but nothing is identifiable from the original data set.

For quick validation that the first and second marginal moments of the synthetic data set are reasonable, you can call PROC MEANS to check that the mean and standard deviation of each variable in the data set is similar to the real data:

```
proc means data=OriginalData mean std;
run;
proc means data=SyntheticData mean std;
run;
```

Variable	Mean	Std Dev
AgeAtStart	44.0730304	8.5678888
Height	64.8262056	3.5862484
Weight	153.0601310	28.9098638
Diastolic	85.4111927	12.9814814
Systolic	136.9874975	23.7544096
Smoking	9.3709069	11.9962262
Cholesterol	227.4107958	44.9230555
Dead	0.3798373	0.4853943
Male	0.4514785	0.4976895

Table 1. PROC MEANS Output for OriginalData

Variable	Mean	Std Dev
AgeAtStart	44.0200000	9.8851792
Height	64.7403782	6.2810369
Weight	152.4900000	36.2417514
Diastolic	85.0000000	17.3868637
Systolic	136.2300000	31.0746429
Smoking	9.7700000	11.3670221
Cholesterol	226.2100000	57.8326207
Dead	0.3900000	0.4902071
Male	0.4600000	0.5009083

Table 2. PROC MEANS Output for SyntheticData

The output in Table 1 and **Error! Reference source not found.** show that the means are close and the standard deviations are similar. Note that standard deviation and covariance use central moments, not raw moments, so their differences can be more noticeable than the mean.

For validation that the interaction between variables is captured, you can call PROC CORR to check that the covariance matrices are similar:

```
proc corr data=OriginalData noprob nocorr cov;
   var AgeAtStart Height Weight Diastolic Systolic Smoking Cholesterol
      Dead Male;
run;
proc corr data=SyntheticData noprob nocorr cov;
   var AgeAtStart Height Weight Diastolic Systolic Smoking Cholesterol
      Dead Male;
run;
```

Covariance Matrix, DF = 5038									
	AgeAtStart	Height	Weight	Diastolic	Systolic	Smoking	Cholesterol	Dead	Male
AgeAtStart	73.408719	-4.145503	22.096838	30.510854	77.544383	-17.723202	105.165309	1.819019	0.001559
Height	-4.145503	12.861177	54.254051	-0.559527	-6.013785	12.370102	-12.676657	0.066870	1.238861
Weight	22.096838	54.254051	835.780226	123.485591	179.819132	31.724020	94.951475	1.953932	6.521676
Diastolic	30.510854	-0.559527	123.485591	168.518858	245.685769	-10.065407	106.762372	1.585942	0.402846
Systolic	77.544383	-6.013785	179.819132	245.685769	564.271975	-26.234346	211.766153	3.535715	0.021724
Smoking	-17.723202	12.370102	31.724020	-10.065407	-26.234346	143.909443	-6.858431	0.547853	2.186818
Cholesterol	105.165309	-12.676657	94.951475	106.762372	211.766153	-6.858431	2018.080916	3.369539	-0.600151
Dead	1.819019	0.066870	1.953932	1.585942	3.535715	0.547853	3.369539	0.235608	0.038482
Male	0.001559	1.238861	6.521676	0.402846	0.021724	2.186818	-0.600151	0.038482	0.247695

Table 3. PROC CORR Output for OriginalData

Covariance Matrix, DF = 99									
	AgeAtStart	Height	Weight	Diastolic	Systolic	Smoking	Cholesterol	Dead	Male
AgeAtStart	97.716768	0.914537	34.495152	37.929293	87.369091	-26.500404	127.854343	0.729495	-0.231515
Height	0.914537	39.451424	52.620450	8.203142	21.104580	-1.530206	60.355475	-0.122426	0.434622
Weight	34.495152	52.620450	1313.464545	160.595960	230.704343	-40.663939	302.421313	-0.657677	2.529899
Diastolic	37.929293	8.203142	160.595960	302.303030	257.070707	-40.595960	198.191919	-0.515152	-0.494949
Systolic	87.369091	21.104580	230.704343	257.070707	965.633434	-81.734444	312.547172	0.303333	-2.561414
Smoking	-26.500404	-1.530206	-40.663939	-40.595960	-81.734444	129.209192	-143.375455	0.555253	1.611919
Cholesterol	127.854343	60.355475	302.421313	198.191919	312.547172	-143.375455	3344.612020	-2.102929	-3.643030
Dead	0.729495	-0.122426	-0.657677	-0.515152	0.303333	0.555253	-2.102929	0.240303	0.020808
Male	-0.231515	0.434622	2.529899	-0.494949	-2.561414	1.611919	-3.643030	0.020808	0.250909

Table 4. PROC CORR Output for SyntheticData

Table 3 and **Error! Reference source not found.** show the covariance matrices. In this case, most of the covariances are on the same order of magnitude, but there are noticeable differences. Using a smaller value for the ALPHA parameter, a larger MINNUMIPCANDS, or a longer MILPMAXTIME usually reduce the differences between the covariance matrices.

LARGER EXAMPLE WITH FITTED MODEL COMPARISON

If the synthetic data are truly similar to the original data, models fit to either data set should score new data similarly. This example splits *OriginalData* into a training data set and a testing data set, creates a synthetic data set based on the training data set, fits a logistic regression model to the training and synthetic data sets, and finally compares the fitted models and how they score against the test data set.

The first step is to randomly partition the data into a training set (70%) and a test set (30%):

```
data Training Testing;
   call streaminit(1);
   set OriginalData ;
```

```
   if RAND('UNIFORM') <= .7 then output Training;
   else output Testing;
run;
```

The next step is to create the synthetic data based on the original data selected for the training set:

```
%GENDATA(INPUTDATA=Training, METADATA=Metadata, NUMCOFORTHREADS=4,
   MOMENTORDER=3, MILPMAXTIME=1200, ALPHA=0.2);
```

Increasing MOMENTORDER to 3 and decreasing ALPHA to 0.2 fits the original distribution more closely and is more precise. The drawback is that it takes much longer to run. Using NUMCOFORTHREADS=4 does help reduce the time by about 40% in this case compared to using NUMCOFORTHREADS=1.

Body mass index (BMI) is generally a better independent predictor than height or weight. Since BMI is a function of height and weight, it is computed and added to the data sets by utilizing the DATA step:

```
data TestingWithBmi;
   set Testing;
   BMI=703*Weight/Height/Height;
run;
data TrainingWithBmi;
   set Training;
   BMI=703*Weight/Height/Height;
run;
data SyntheticDataWithBmi;
   set SyntheticData;
   BMI=703*Weight/Height/Height;
run;
```

Now you can use PROC LOGISTIC to fit a logistic regression and score against the original data:

```
proc logistic data=TrainingWithBmi;
   model Dead = Male AgeAtStart Diastolic Systolic Smoking Cholesterol BMI
      / outroc=troc;
   score data=TestingWithBmi out=valpred outroc=vroc;
run;
proc logistic data=SyntheticDataWithBmi;
   model Dead = Male AgeAtStart Diastolic Systolic Smoking Cholesterol BMI
      / outroc=troc;
   score data=TestingWithBmi out=valpred outroc=vroc;
run;
```

A large amount of output is produced, but only some of the tables and figures are included here. Table 5 and Table 6 display the odds ratio estimates and the confidence limits for each variable for *TrainingWithBmi* and *SyntheticDataWithBmi,* respectively. The ratios of four of the seven variables for the synthetic data fall within the 95% confidence intervals. The others are still relatively close.

Figure 1 and **Error! Reference source not found.** show the ROC curves for the models scored against the test data. The area under the curve (AUC) included in the figures is a measure of the model's specificity and sensitivity. A value of 1 would be perfect, and a value of 0.5 would mean it is as good as randomly guessing. While the values of AUC for the fitted models were 0.8069 for the training data and 0.7141 for the synthetic data, the values of the models applied to the test data were very close to each other at 0.7800 and 0.7768. This indicates that even though there are some clear differences between the fitted models, they performed similarly against new data from the original distribution.

Odds Ratio Estimates			
Effect	**Point Estimate**	**95% Wald Confidence Limits**	
Male	0.521	0.437	0.621
AgeAtStart	0.883	0.874	0.894
Diastolic	0.992	0.982	1.003
Systolic	0.985	0.979	0.991
Smoking	0.970	0.963	0.977
Cholesterol	0.999	0.997	1.000
BMI	0.998	0.978	1.018

Table 5. Odds Ratio Estimates for TrainingWithBmi

Odds Ratio Estimates			
Effect	**Point Estimate**	**95% Wald Confidence Limits**	
Male	0.690	0.595	0.799
AgeAtStart	0.933	0.925	0.941
Diastolic	0.992	0.987	0.998
Systolic	0.997	0.994	1.000
Smoking	0.986	0.982	0.991
Cholesterol	0.999	0.998	1.000
BMI	0.992	0.982	1.003

Table 6. Odds Ratio Estimates for SyntheticDataWithBmi

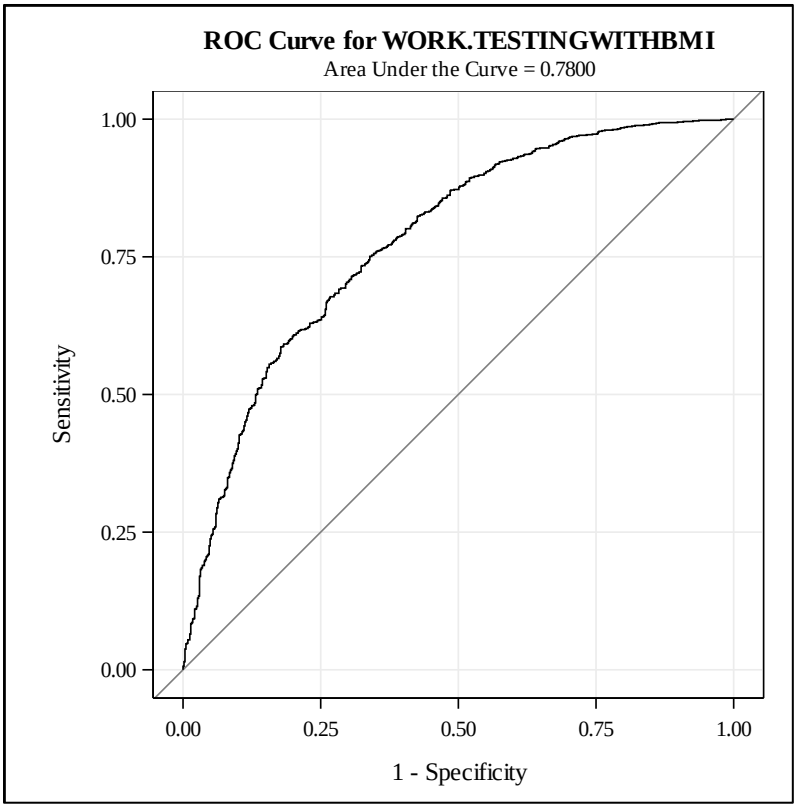

Figure 1. ROC Curve. Model derived in TrainingWithBmi and tested in TestingWithBmi

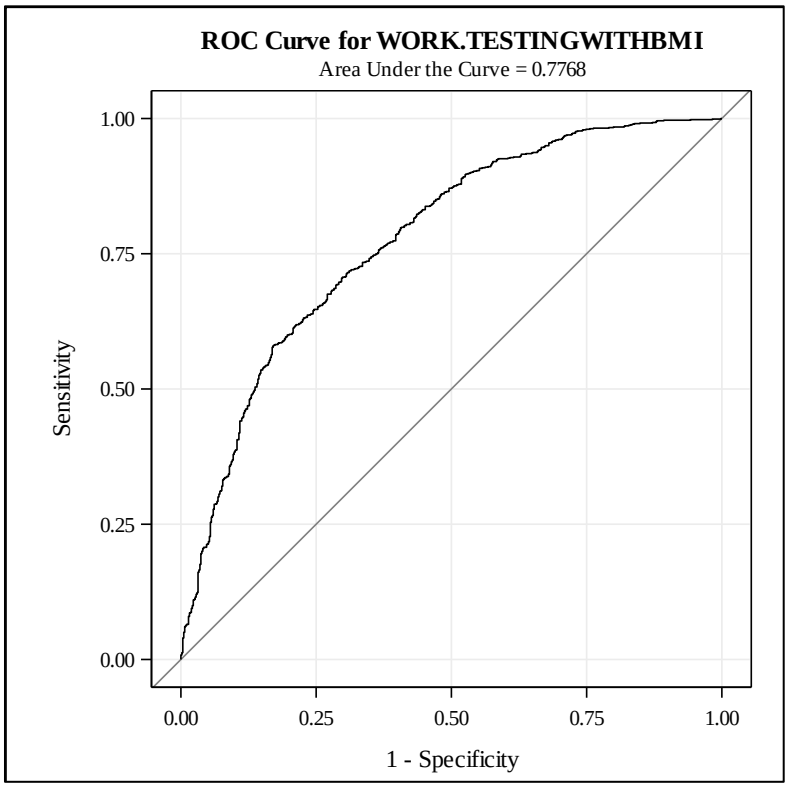

Figure 2. ROC Curve. Model derived in SyntheticDataWithBmi and tested in TestingWithBmi

This comparison is only for one instance and one model, but demonstrates that the synthetic data produced by this implementation can be useful substitutes for the real data. An in-depth analysis of the performance and a comparison to other synthetic data methods has been previously published (Bogle 2016).

POTENTIAL FOR CUSTOMIZATION

Numerous portions of this macro allow for customization. The lower and upper bounds on the moments can be modified to more accurately reflect the desired precision. For example, the moments could be specified to have a lower and upper bound of within one percent of the point estimate of the corresponding moment from the real data, or could be manually specified. Additionally, if only certain moments are known to be important during the matching process, these can be the sole moments specified. To emphasize lower order moments, for example, tighter bounds could be placed on first and second order moments and wider bounds on third and fourth order moments.

There are opportunities to reduce the number of unused LP candidate observations by specifying the sampling technique used to generate values for each variable. For example, if a variable has a distribution that is thought to be very different from uniform, a different distribution could be specified based on the empirical distribution or literature. Using a multivariate distribution could help find synthetic observations that better match mixed moments, if these relationships are well-known.

If multiple synthetic data sets are desired for one original data set, the preliminary step would only need to be run once. A loop could be put around the LP and IP steps, with care taken to prevent data sets from being overwritten.

CONCLUSION

This paper demonstrates a method for producing synthetic data using the features of the OPTMODEL procedure in SAS/OR. It leverages several strengths of PROC OPTMODEL, including the ability to easily read from and write to data sets, use statistical functions, easily run solvers in parallel using the COFOR loop, use dual information to compute reduced costs for column generation, and apply other SAS procedures to data sets produced.

Our approach to generating synthetic data can be used to create many loosely matched data sets quickly. If maintaining statistical properties is critical and with enough lead time, it can be used to produce a single data set that very closely matches the moments of the real data without disclosing any of the real data's observations. Properly specified synthetic data can streamline the research process by removing some barriers to data access and therefore is a potential mechanism to increase the utility of sensitive and expensive data. Crowdsourcing solutions through platforms like Kaggle is also easier to utilize for problems that would normally have privacy restrictions.

REFERENCES

Bogle, B.M. and S. Mehrotra. 2016. "A Moment Matching Approach for Generating Synthetic Data." *Big Data,* 4(3):160–178.

Desaulniers, A. and J. Desrosiers and M.M. Solomon. 2005. *Column Generation*. NewYork, NY: Springer US.

Schrijver, A. 1998. *A Theory of Linear and Integer Programming*. Hoboken, NJ: John Wiley & Sons.

ACKNOWLEDGMENTS

A special thanks to Rob Pratt for his help with improving the PROC OPTMODEL code and reviewing this manuscript.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Brittany Bogle
bbogle@email.unc.edu