Randomly Split SAS Data Set Exactly According to a Given Probability Vector

Liang Xie
Reliant Energy, NRG

Aug 20, 2009

Abstract

In this paper, we examine a fast method to randomly split SAS data set into N pieces exactly according to a given probability vector. The method, which scans the data only twice at the worst case, is an extension of the K/N algorithm extensively discussed on SAS-L archieve. We first discuss the mathetmatical rationale behind this algorithm, then we demonstrate the macro implementation utilizing array and hash table. Lastly, we compare our method to method utilizing SURVEYSELECT and discuss their comparative advantage and disadvantage.

1 Introduction

[Introduction] Once upon a time in my career, one of the SAS programmers in data management team told me that she had difficulties in splitting the population data exactly according to the spliting probability vector I gave to her, and I have to come up with this program to help her and her team. It turned out that many SAS programmers were still using simple strategies to split SAS file based on given splitting probability. These strategies includes:

1. Append a uniform random variable to the original data set and split the data according to the uniform random variable;
2. Use `int(ranuni(&seed)*&TotalPieces)` approach to split the table on the fly based on returned integer value;
3. Iteratively apply `PROC SURVEYSELECT` to the original data and non-selection parts after each iteration;

All of these methods have their disadvantages, either in terms of final statistical property of the sample or efficiency or both. Method 1 is inefficient and can't guarantee the splitting probability as given one even for very large data set. Method 2 has the same problem even though it avoids the step to append a random variable. Both approaches can't guarantee

rigourous statistical property of the final sample, and implementation becomes more complex when strata needs to be considered. Method 3 has SAS-backed rigorousness when used appropriately, but each run only outputs one split piece at a time, so that the efficiency decrease dramatically as the number of splits increases. For example, with ten splits of equal probability, SAS has to run through $5.5+1$ times observations of the original table. In general, when a given probability vector p of k-by-1, it is required to run through $1+\sum_{i=1}^{k}\sum_{j=i}^{k} p_j$ times observations of original table, which is $2*k/3$ asymptotically. Given tight timeline of project and business requirements, an efficient approach that holds sound statistical property is necessary. I employed the K/N algorithm demonstrated by SAS [1], and discussed extensively on SAS-L archieves, search for threads "Random Selection: Anything More Elegant", or see Whittington [2], Autom [3] for details. This algorithms relies on the conditional probability for subsequent sampling without replacement.

In our project, we are not simply implementing this algorithm, but extending it to accommendate data with M stratas, and it is required that each sample has the same strata ratios as in the original data. For example, in any of the output sample pieces, the ratio of MALE and FEMALE of strata GENDER must be the same as that in the original data.

2 The Algorithm

The K/N algorithm is used in K out of N observations random sampling without replacement. The idea behind this algorithm is that the marginal probability of selection of the $(S+1)^{th}$ observation, conditional on the fact that S observations have been selected in the previous M observations is still K/N. A nicely presented proof can be found at [3].

But randomly sample K out of N observations without replacement can be regarded as randomly split the data into predefined two pieces , one is the selected sample with probability K/N, the other is the left over with probability $(K-1)/N$. The decision boundry is determined by comparing a uniform random variable to the constantly updated conditional selection probability at each step. Note, however, that the selection probability for left over part is affected, too, when the selection probability for selected part is updated. Because probabilities sum to 1, when the selection probability for selected part is updated with $(K-1)/(N-1)$, the counter part probability for left over part is also updated as $(N-K)/(N-1)$, which can be understood as the updating process for selecting $N-K$ out of N observations as well. When we have more than two pieces, say M, to output to, the decision has to be made at $M-1$ boundries for the uniform random variable. For example, if it is required to randomly split original data into M pieces, with probability $p_i,\ i \in 1:M, \sum_i^M p_i = 1$, it is the same as randomly sample $N*p_i$ observations out of N for piece i independently across pieces, so that other pieces $j,\ j \neq i$ can be collectively regarded as the left over part. Therefore apply K/N algorithm at each step is validated for random splitting, simply change the boundry conditions accordingly due to identification of different M pieces.

It is desired to go one step further to ensure that exact strata ratio is also inherited in output samples. A naive way to do so is to apply the K/N algorithm to each strata value subset. For example, suppose the original data has a strata X with z unique different values. This is without generality because if there are more than one stratum, we can simply use the combinations of different values of all strata and regard this combination as one stratum. Then we first apply the K/N algorithm to each of the z subsets, and then combine the data. Taking a simple example, suppose we have a data with strata variable GENDER having two unique values: MALE and FEMALE and we want to split it into M pieces randomly. We first use the simple K/N algorithm to the MALE subset, get M pieces for MALE; and then conduct the same operation to the FEMALE subset, getting another set of M pieces of sample; then these pairs of M samples are combined accordingly at the final step. While this approach is absolutely legal, it is very inefficient, but it does shed light on where we can make improvement.

When the original data are firstly splitted into subset by strata values and then apply K/N algorithm to each one, we are using their conditional probability, that is conditional probability of selection observation S that belongs to stratum value X_k is the same as calculated by original K/N algorithm, but the marginal selection probability across the whole data set should be adjusted by the current proportion of stratum X_k in the remaining portion. This in turn implies that we can transform the random splitting with strata into the original simple splitting problem by treating each combination of Stratum Value and Piece as a new unique piece, where the splitting probability is the product of Stratum Value proportion and splitting probability. So that randomly splitting into M pieces with z stratum values becames randomly splitting into $M * z$ pieces, and we simply update the splitting probability at each observation based on its stratum value.

3 Algorithm Implementation

Because the more complex problem can be transformed into the simplest case, we first demonstrate the implementation in problems without strata constraints. The key idea is updating the conditional probability. At the 1^{st} observation, the selection probability is K_i/N for piece i, $i \in 1 : M$. Then we generate a uniform random variable u, and output to sample i if $\sum_0^{i-1} p_j \leq u \leq \sum_0^i p_j,\ p_0 = 0$, and the conditional selection probability for piece i becomes $(K_i - 1)/(N - 1)$, and for pieces j, $j \neq i$, their conditional selection probabilities becomes $K_j/(N - 1)$. This simple algorithm can be implemented as the following code:

```
data New;
     set original  nobs=nobs0;
     array _P{&M} _temporary_;
     array _F{&M} _temporary_;
     array _K{&M} _temporary_;
     if _n_=1 then do;
        _temp_=0;
```

```
        do i=1 to nobs1;
           set Probability  nobs=nobs1;
           _P[i]=Prob;
           if i=1 then _F[i]=_P[i];                  .....................(1)
           else _F[i]=_F[i-1]+_P[i];
           if i<&M then do;
              _K[i]=int(nobs0*_P[i]); _temp_+_K[i]; .....................(2)
           else _K[i]=nobs0-_temp_;
        end;
     end;
     u=ranuni(&seed);
     notfound=1; j=1;
     do while (j<=&M & notfound);
        if r<=_F[j] & _K[j]>0 then do;              .....................(3)
           BLOCK=j; notfound=0; _K[BLOCK]-1;
        end;
        else do;
           j+1;
        end;
     end;
     do j=1 to &M;
        _P[j]=_K[j]/(nobs1-_n_);                    .....................(4)
        if j=1 then _F[j]=_P[j];
        else if j<&M then _F[j]=_F[j-1]+_P[j];
        else _F[j]=1-_F[j-1];
     end;
     drop j notfound u;
run;
```

In step (1), we construct the splitting boundries based on initial marginal probability vector, then in step (2), required sample size in each splitting piece is calculated. Step (3) decides which piece the observation should be sent to and step (4) updates the conditional selection probability based on current output.

Extending this code to accommendate multiple strata, we simply expand this one dimensional arrays to two dimensions where the first dimension corresponding to the strata while the second dimension corresponding to splitting probability vector. That is, conditinal on the strata value, corresponding row in the probability matrix `_P` and frequency matrix `_F` are updated. For example, instead of

```
array _P{&M} _temporary_;
array _F{&M} _temporary_;
array _K{&M} _temporary_;
```

we use 2-dimensional arrays:

```
array _Pmat{1:&ncomb, 1:&M} _temporary_;
array _Fmat{1:&ncomb, 1:&M} _temporary_;
array _Kmat{1:&ncomb, 1:&M} _temporary_;
  array _Nmat{1:&ncomb, 0:%eval(&M+1)} _temporary_;
```

where &ncomb is the number of unique strata values. The extra matrix _Nmat is used to store the number of required observations for each strata value and splitting piece combination. It has extra columns than other matrices to store agrregate information for each strata and this information is to ensure the final output probability will be exact. Since the rows corresponds to strata value order, for each observation, we can quickly locate which row of these matrices corresponds to its strata value by looking up strata value index, which can be accomplished by a hash table:

```
  %if &withcontrol=1 %then %do;
      declare hash h();
      h.defineKey(%str("&control_key"));
      h.defineData("_index_");
      h.defineDone();
      _index_=1;
      do while (^eof);
         set  _freq_  end=eof;
         rc=h.add();
         cn=0;
         _Nmat[_index_, 0]=COUNT; _Nmat[_index_, %eval(&n+1)]=0;
         do j=1 to &n;
            if j<&n then do;
              _Nmat[_index_, j]=round(_P[j]*COUNT*PERCENT, 1);
              cn+_Nmat[_index_, j];
            end;
            else _Nmat[_index_, j]=COUNT-cn;
            _Xmat[_index_, j]=_Nmat[_index_, j];
            _Kmat[_index_, j]=0;
         end;
         _index_+1;
      end;
 %end;
```

In this step, the macro variable &withcontrol indicate if strata presents, and if it is the case, the program define a hash table h with satelite data _index_ indicating the rows of the matrices. Then it reads in the frequency table generated by PROC FREQ, assign each unique combination of strata values an index. In this way, we build up a quick look up table for the strata values in corresponding matrices. Hence, in the probability updating step, we simply update the probability of corresponding rows:

```
  _Pmat[_index_, BLOCK]
= _Xmat[_index_, BLOCK]/(_Nmat[_index_, 0]-_Nmat[_index_, %eval(&M+1)]);
```

4 Some Experiments and Conclusion

We generate a synthetic data to see if the program works. The testing data set has two strata: "TDSP" and "VS", we expect the output pieces have the same strata ratios as the original data. Macro %split can be found in the appendix.

```
data test0;
```

Desired Output								
PROC FREQ output			_prob_1	_prob_2	_prob_3	_prob_4	_prob_5	Total
COUNT	PERCENT	_index_	0.1485905	0.1485905	0.2324283	0.2351954	0.2351954	
30083	3.7756	1	4470	4470	6992	7075	7076	30083
69544	8.7282	2	10334	10334	16164	16356	16356	69544
59849	7.5114	3	8893	8893	13911	14076	14076	59849
138782	17.418	4	20622	20622	32257	32641	32640	138782
59748	7.4987	5	8878	8878	13887	14052	14053	59748
139001	17.4455	6	20654	20654	32308	32692	32693	139001
90134	11.3124	7	13393	13393	20950	21199	21199	90134
209632	26.3101	8	31149	31149	48724	49304	49306	209632
Total			118393	118393	185193	187395	187399	796773

%split output Count								
Sum of COUNT			_BLOCK					Total
TDSP	VS	_index_	1	2	3	4	5	
0	H	1	4470	4470	6992	7075	7076	30083
0	M	2	10334	10334	16164	16356	16356	69544
1	H	3	8893	8893	13911	14076	14076	59849
1	M	4	20622	20622	32257	32641	32640	138782
2	H	5	8878	8878	13887	14052	14053	59748
2	M	6	20654	20654	32308	32692	32693	139001
3	H	7	13393	13393	20950	21199	21199	90134
3	M	8	31149	31149	48724	49304	49306	209632
Total			118393	118393	185193	187395	187399	796773
Difference			0	0	0	0	0	0

Figure 4.1: Desired Output and Actual Output

```
     do ID=1 to 796773;
        TDSP=min(3, round(ranuni(8976)*4));
        if ranuni(93745)<0.3 then VS='H'; else VS='M';
      output;
     end;
run;

options mprint mlogic;

%let seed=99999;
%let probvector=0.1485905 0.1485905 0.2324283 0.2351954 0.2351954;
%let control_vars=TDSP  VS;
%let in_dsn=test0;
%let out_dsn=test_out;
%let sort=NEST;
%split(&seed, &probvector, &control_vars, &in_dsn, out_dsn=&out_dsn);

options nomprint nomlogic;

;;
```

Figure 4.1 shows the result follows the splitting probability exactly:

In this paper, we demonstrate an efficient way to randomly split a SAS data set into mutliple pieces exactly according a given splitting probability vector and the extension of original K/N algorithm allows multiple strata and ensures the strata ratio in output file will be the same as the original file. The full code of the macro can be found in the appendix.

5 Reference

1. SAS Technical Support, *Sample 24722: Simple random sample without replacement* Method 3., http://support.sas.com/kb/24/722.html
2. Whittington, John, http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9909C&L=sas-l&P=R6540&D=0&H=0&O=T&T=1
3. Autom, Tim, http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9909C&L=sas-l&P=R10451&D=0&H=0&O=T&T=1

6 Contact Information

Liang Xie
Reliant Energy
1000 Main St
Houston, TX 77081

Work phone: 713-497-6908
E-mail: xie1978@yahoo.com
Web: www.linkedin.com/in/liangxie
Blog: sas-programming.blogspot.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration. Other brand and product names are trademarks of their respective companies.

7 Appendix

```
/************************************************/
%macro split(seed,
             probvector,
             control_vars,
             in_dsn,
             out_dsn=_out_dsn_,
             sort=nest);
%let blank=%str( );
%let n=1; %let nv=1;
%let probs=&probvector;

%let dsid=%sysfunc(open(&in_dsn));
%if  (&dsid=0) %then %do;
     %put %sysfunc(sysmsg());
   %put Stop Processing due to Errors;
   %goto exit;
%end;
%else %do;
    %let nobs=%sysfunc(attrn(&dsid, NLOBS));
  %let dsid=%sysfunc(close(&dsid));
```

```
%end;

%if &out_dsn=&blank %then %let &out_dsn=&in_dsn;
%if &seed=&blank %then %let seed=0;

%let pos=%scan(&probs,&n,&blank);
%do %while(&pos ne &blank);
    %let n=%eval(&n+1);
    %let pos=%scan(&probs,&n,&blank);
%end;
%let n=%eval(&n-1);

%let pos=%scan(&control_vars,&nv,&blank);
%let control_vars_x=&pos;
%let control_key=%str(&pos);
%let control_find=%str(key:&pos);
%do %while(&pos ne &blank);
    %put &pos;
    %let nv=%eval(&nv+1);
    %let pos=%scan(&control_vars,&nv,&blank);
  %if &pos^=&blank %then %do;
      %let control_vars_x=%str(&control_vars_x * &pos);
        %let control_key=%str(&pos%", %"&control_key);
      %let control_find=%str(key:&pos, &control_find);
  %end;
%end;
%let nv=%eval(&nv-1);
%put nv=&nv;
%if %eval(&nv>0) %then %do;
    %if %upcase(&sort)=NEST %then %do;
        proc sort data=&in_dsn out=_in_dsn_;
             by &control_vars;
        run;
    %end;
    %else %if %upcase(&sort)=SERP %then %do;
        ods select none;
          proc surveyselect data=&in_dsn
                            samprate=1.0
                            out=_in_dsn_
                            sort=&sort
                            method=sys;
           control  &control_vars;
        run;
      ods select all;
    %end;
    %else %do;
          %put Specify only NEST or SERP for sort statement;
        %goto exit;
    %end;
  %let in_dsn=_in_dsn_;
```

```
  proc freq data=_in_dsn_ noprint;
       tables &control_vars_x /missing out=_freq_;
  run;
  %let withcontrol=1;
  %let dsid=%sysfunc(open(_freq_));
  %let ncomb=%sysfunc(attrn(&dsid, NOBS));
  %let dsid=%sysfunc(close(&dsid));
%end;
%else %do;
    %let withcontrol=0;
  %let ncomb=1;
%end;

data probs;
     %do i=1 %to &n;
       _prob_&i=%scan(&probs, &i, &blank);
     %end;
   output;
run;

%let i=&n;
data &out_dsn  Nmat(keep=_n:);
     array _P{*}  _prob_1-_prob_&n;
   array _Pmat{1:&ncomb, 1:&n} _temporary_;
   array _Fmat{1:&ncomb, 1:&n} _temporary_;
   array _Xmat{1:&ncomb, 1:&n} _temporary_;
   array _Kmat{1:&ncomb, 1:&n} _temporary_;
   array _Nmat{1:&ncomb, 0:%eval(&n+1)} _temporary_;
   if _n_=1 then do;
      set probs;
    do i=1 to &ncomb;
       do j=1 to &n;
          _Pmat[i, j]=_P[j];
        if j=1 then _Fmat[i, j]=_P[j];
        else _Fmat[i, j]=_Fmat[i, j-1]+_P[j];
       end;
       do j=1 to &n;
              _Fmat[i, j]=_Fmat[i, j]/_Fmat[i, &n];
       end;
    end;

        %if &withcontrol=1 %then %do;
        declare hash h();
      h.defineKey(%str("&control_key"));
      h.defineData("_index_");
      h.defineDone();
        _index_=1;
            do while (^eof);
         set  _freq_  end=eof;
```

```
        rc=h.add();
        cn=0;
        _Nmat[_index_, 0]=COUNT; _Nmat[_index_, %eval(&n+1)]=0;
        do j=1 to &n;
           if j<&n then do;
                    _Nmat[_index_, j]=round(_P[j]*COUNT*PERCENT, 1);
          cn+_Nmat[_index_, j];
         end;
         else _Nmat[_index_, j]=COUNT-cn;
         _Xmat[_index_, j]=_Nmat[_index_, j];
         _Kmat[_index_, j]=0;
        end;
        _index_+1;
           end;
       %end;
   %else %do;
           _index_=1; cn=0;
     do j=1 to &n;
        if j<&n then do;
                 _Nmat[_index_, j]=round(_P[j]*&nobs, 1);
         cn+_Nmat[_index_, j];
        end;
        else _Nmat[_index_, j]=&nobs-cn;
        _Xmat[_index_, j]=_Nmat[_index_, j];
        _Kmat[_index_, j]=0;
    end;
  %end;
end;
set &in_dsn  nobs=ntotal end=eof;
      r=ranuni(&seed);
    %if &withcontrol=1 %then %do;
        rc=h.find(&control_find);
    %end;
    %else %do;
       _index_=1;
    %end;
    notfound=1;  j=1;
    do while (j<=&n & notfound);
         if r<=_Fmat[_index_, j] & _Kmat[_index_, j]<_Nmat[_index_,j] then do;
        BLOCK=j; notfound=0; _Kmat[_index_, BLOCK]+1;
     end;
     else do;
            j+1;
     end;
    end;

      _Xmat[_index_, BLOCK]=_Nmat[_index_, BLOCK]-_Kmat[_index_, BLOCK];
      _Nmat[_index_, %eval(&n+1)]+1;

    if ^(eof ) then do;
```

```
           _Pmat[_index_, BLOCK]=_Xmat[_index_, BLOCK]/
                                (_Nmat[_index_, 0]-_Nmat[_index_, %eval(&n+1)]);
       do j=1 to &n;
              if j=1 then _Fmat[_index_, j]=_Pmat[_index_, j];
              else _Fmat[_index_, j]=_Fmat[_index_, j-1]+_Pmat[_index_, j];
       end;
       do j=1 to &n;
          _Fmat[_index_, j]=_Fmat[_index_, j]/_Fmat[_index_, &n];
       end;
    end;
    drop _prob:  _n_:  i r;
run;

%exit:
%mend;



/***************/

%macro wrap;
%let n=1; %let nv=1; %let blank=%str( );
%let control_vars=TDSP  VS  DWELLING_TYPE_CD;
%let pos=%scan(&control_vars,&nv,&blank);
%let control_vars_x=&pos;
%let control_key=%str(&pos);
%let control_find=%str(key:&pos);
%do %while(&pos ne &blank);
    %put &pos;
    %let nv=%eval(&nv+1);
    %let pos=%scan(&control_vars,&nv,&blank);
    %if &pos^=&blank %then %do;
        %let control_vars_x=%str(&control_vars_x * &pos);
        %let control_key=%str(&pos%", %"&control_key);
        %let control_find=%str(key:&pos, &control_find);
    %end;
%end;
%let nv=%trim(%eval(&nv-1));
%put nv=&nv;
%put pos: &pos;
%put control_vars_x: %str("&control_vars_x");
%put control_key: %str("&control_key");
%put control_find: &control_find;
%mend;

%wrap;


options mprint mlogic;
```

```
%let seed=986532147;
%let probvector=0.1485905 0.1485905 0.2324283 0.2351954 0.2351954;
%let control_vars=TDSP  VS;
%let in_dsn=test0;
%let out_dsn=test_out;
%let sort=NEST;
%split(&seed, &probvector, &control_vars, &in_dsn, out_dsn=&out_dsn);

options nomprint nomlogic;

;;
```