Paper 072-30

Automating Predictive Analysis to Predict Medicare Fraud

Christine L. Warner, SRA International, Fairfax, Virginia

ABSTRACT

Medicare fraud is a reality despite efforts to prevent and detect fraud and abuse. "On February 21, 2002, the HHS-OIG reported its finding that of the $191.8 billion in such [Medicare] claims paid in 2001, 6.3 percent—amounting to **$12.1 billion**— should not have been paid due to erroneous billing or payment, inadequate provider documentation of services to back up the claims and/or outright fraud (National Health Care Anti-Fraud Association)." In response to this growing problem, our team of data miners was tasked with identifying Medicare fraud using SAS software. One approach to detecting potential Medicare fraud is to identify providers who make significantly more money than their peers, a process that requires predictive analysis.

We decided to use SAS to predict how much each provider should be making based on a number of factors such as: patient volume, what they do (procedures), where they work (place of service), and who they treat (beneficiary factors). Once we had a reliable model, we then identified the providers who made significantly more than their predicted allowed amount, and flagged them for further research.

We had two stipulations on our solution: 1) our client did not have Enterprise Miner and therefore wanted the solution created using Base SAS and SAS/Stat, and 2) our client wanted the solution to be a "push-button" solution that they could run for any physician specialty group for a given time period. *We* also wanted the model to be push-button because the development of a reliable model is an incredibly iterative process for which we wanted to avoid tedious hard-coding in the process. These two restrictions helped formulate our plan to automate as many processes in the predictive analysis as possible.

Out client wanted a regression model that could be run for each provider specialty code. (Specialty codes describe the provider's specialty, such as "Internal Medicine", or "Oncology"). Because factors differ so widely among specialty groups, our proposed solution had to be **a dynamic regression model that created and selected its own variables**. Some of the sub-tasks we automated were:

- ✓ Variable creation based upon changing specialty code
- ✓ Variable selection based upon percentage of missing values and dollar amount
- ✓ Identifying and removing outliers for model training
- ✓ Creating a dynamic string of independent variable names to feed into the model
- ✓ Using PROC REG and PROC SCORE to run the regression analysis
- ✓ Conducting cluster analysis prior to model development to identify sub-groups that may exist, and developing a model for each cluster

This paper discusses one approach to automating predictive analysis.

INTRODUCTION

Our automated, dynamic approach to predictive analysis was born from two circumstances: 1) our client's software, and 2) our laziness. Our client did not have Enterprise Miner, which offers the most automated approach to predictive analysis; therefore, we used SAS and SAS/Stat to complete the task. One of the sub-tasks we automated that Enterprise Miner currently offers is a variable selection process. In Enterprise Miner, this selection node allows users to set a threshold for variable selection based upon the percentage of missing values in that field. We replicated this variable selection node using Base SAS code by setting the threshold value as a global macro variable.

Our approach was also born from our laziness: we wanted to avoid hard-coding as much as possible! We were tasked to investigate seven specialty codes, each having potentially several hundred different independent factors. Faced with hard-coding each procedure code, each place of service, and each diagnosis code as an independent variable, we sought another approach: automation!

During the model development process, we automated many of the following processes, each of which is addressed in the following pages.

- Variable creation
- Variable selection
- Outlier removal
- Creation of a dynamic list of variables
- Model development
- Cluster analysis
- Flagging the final outliers
- Model comparison

VARIABLE CREATION

Because the factors associated with each physician specialty are so different, we had to come up with a way to automate variable creation for each specialty code and date range given. There were three main factors that changed for each specialty:

- Procedure: the type of procedures performed on the beneficiary
- Place of service: where the physician performed the procedure
- Diagnosis: the diagnosis code for the beneficiary

For each of these factors, the program builds many variables that describe the percentage of business (allowed amount) the physician performed in each place of service, for each procedure, and for each diagnosis code.

PROCEDURE CODES

One of our goals was to include in the model a factor that represented WHAT the physician did: the procedure code. The procedure code we used is the Healthcare Common Procedure Coding System (HCPCS) 5-digit code. We thought of creating a variable that contained the percentage of business (dollars) that each provider billed to each procedure. However, there are approximately 9,330 procedures (Centers for Medicare and Medicaid Services, 2005), and including this many variables in a model was not a possibility. So we condensed our variables to include just the first couple of digits of the procedure code. For example, the variable PROC1_7 represents the percent of a provider's business billed to all procedures starting with "7", of which there are 685 procedure codes. Likewise, PROC2_78 represents the percent of a provider's business billed to all procedures starting with the 2 digits "78", of which there are 145. We were really trying to circumvent the problem of hard-coding all of the possible PROC1, PROC2, and PROC3 combinations for each physician specialty code, of which there could be hundreds.

To do this, we relied on PROC TRANSPOSE:

```
proc transpose data=proc1_pct out=procedures prefix=proc1_ ;
    by upin_id;
    id proc1;
    var proc1_pct ;
run;
```

This code changed the database shape from vertical to horizontal:

VERTICAL LAYOUT:

PHYSICIAN ID:	PROC1:	PROC1_PCT:
0001	1	10%
0001	2	15%
0001	3	25%
0001	5	40%
0001	9	10%

HORIZONTAL LAYOUT:

PHYSICIAN ID:	PROC1-1	PROC1-2	PROC1-3	PROC1-5	PROC1-9
0001	10%	15%	25%	40%	10%

Using the PROC TRANSPOSE created several new variables in the process (5 in this example).

Then we needed to convert missing numeric values to zero to avoid any errors in the regression analysis. To do this, I used the following code which I found on the SAS technical support website:

```
data proc1(drop= _name_ _label_);
    set procedures;
    array testmiss(*) _numeric_;
    do i = 1 to dim(testmiss);
        if testmiss(i)=. then testmiss(i)=0;
    end;
run;
```

We repeated this process for PROC1 (first digit of procedure), PROC2 (first 2 digits), and PROC3 (first 3 digits).

PLACE OF SERVICE

There are 27 different place-of-service codes in the Medicare billing system. These are 2-digit codes that range from "11" (Office), to "41" (Ambulance) to "99" (Unlisted). We employed exactly the same logic as we did in creating the procedure code fields, meaning we again relied on PROC TRANSPOSE to create the 27 different variables. Each provider of course will not have worked in all 27 places of service, so we again had to fill the missing values with zeros to avoid errors in the regression and cluster analysis.

DIAGNOSIS CODE

The diagnosis code is a 5-digit code describing the specific health condition of the patient. Some examples of the diagnosis code are:

Code	Description
524.30	Unspecified anomaly of tooth position
524.31	Crowding of teeth
524.32	Excessive spacing of teeth
524.33	Horizontal displacement of teeth
524.34	Vertical displacement of teeth

There are too many diagnosis codes to include each one as a separate variable. To handle this problem, we condensed the diagnosis codes into their first two digits, as we did in the procedure code solution. For example, the variable DIAG_52 represents the percentage of business a provider billed to all diagnosis codes that start with "52". Generally, all diagnosis codes starting with "52" pertain to dental diagnoses. Condensing the variables in this manner retained important categorization of the data without having to address every single diagnosis code.

OTHER VARIABLES

The other independent variables in the model were:

- Total number of beneficiaries for a provider (for a given date range)
- Total number of procedures that a provider performed in the date range provided
- Percent of beneficiaries who are high-risk patients (more seriously ill)
- Percent of beneficiaries who are male / female
- Percent of beneficiaries who are above age 65 / below age 65
- Whether or not the provider works in a Metropolitan Statistical Area (MSA)

These variables did not need to be dynamically created; their algorithms remained the same for each specialty code.

The dependent variable in the model was usually the log Base 10 of the allowed amount for a given provider. We changed this for some model runs to the raw allowed amount for comparison purposes.

VARIABLE SELECTION

Often the number of variables created would be in the hundreds, as some specialty codes such as "general practitioner" involved many procedures and many diagnoses. This necessitated the creation of an **automatic variable-selection function**, which limited the number of independent variables input in the model. We decided to limit the number of variables input in the model by accepting variables that had a percentage of non-missing values that was greater than a user-defined threshold percentage. The default value for this threshold was set at 50%, meaning that only variables that were at least 50% populated were included in the model. You could set this threshold at any percentage, however.

The SAS code to achieve the automatic variable selection is included below. The incoming data set has 1 record per physician, and many variables that describe how much of their business is performed in each procedure-group that started with the same first 2 digits. For example, the variable PROC2_99 describes how much business a doctor charged to all procedures starting with "99", which is the code generally used to bill an office visit. Likewise, the variable PROC2_J9 describes the percentage of business a doctor charged to all procedures starting with "J9", which are chemotherapy procedures.

First, define your population threshold using the macro variable POPMIN. We used 50%, but you could change this to any percentage. To define each variable's population statistic, we divide the number of non-missing values of that variable by the total number of observations in the data set (which is the total number of physicians). The number of non-missing values is calculated using a PROC MEANS with the n= option, and the number of observations in the data set is calculated using a NOBS option on a SET statement.

```
/* set population threshold: */

%let popmin = .50;

/* calculate number of non-missing values for each variable: */

proc means data=proc2_pct noprint;
    by proc2;
    var proc2_pct;
    output out=stats n=n ;
run;

/* calculate total number of observations: */

data _null_;
    set unique nobs=numobs;
    call symput('numupins',numobs);
run;

/* calculate each variable's population statistic, and keep it in the data set
only if it is greater than the specified threshold: */

data filter;
    set stats;
    if n/&numupins gt &popmin ;
run;
```

The resulting data set FILTER will only include variables that are at least 50% populated. This is one way of ensuring your data set will only include variables that have a good chance of impacting your model.

We also limited the number of variables included in the regression model by the total dollar amount they contributed. This also was treated as a user parameter, global Macro variable, which was initially set at

$2,000. This threshold amount meant that a variable would only be included in the model if, for all Medicare providers, the total amount was greater than or equal to $2,000. This was just another way to include only variables that would make a significant contribution to the model.

First, we identified the variables that totaled less than $2,000, using the following SAS code:

```
data deletevars;
    retain string;
    LENGTH VAR $10 STRING $4000;
    set temp;
        by dummy;
    if first.dummy then string=' ';
    var= compress('proc3_'||proc3);
    if proc3_amt lt 2000;
    string = trim(string)||' '||compress(var);
    len = length(string);
    if last.dummy then output; /* then keep */
run;
```

Then we created a dynamic string of variables and assigned the list to a macro variable called DELSTRING using the CALL SYMPUT statement:

```
data delete_string;
    retain delstring ;
    length delstring $8000;
    set delete_strings;
        by dummy;
    if first.dummy then delstring=' ';
    delstring= trim(delstring)||' '||trim(string);
    if last.dummy;
    call symput('DELSTRING',delstring);
run;
```

Then, later in the program, we used the DELSTRING macro variable in a DROP = statement to delete the variables that totaled less than $2,000:

```
Data final (drop = &delstring);
   Set temp;
Run;
```

(SAS actually can handle hundreds of independent variables; however, we wanted to limit the number of variables fed into the model for efficiency's sake. Also, we wanted to later run the same dataset in other packages such as SAS Enterprise Miner and SPSS Clementine to verify the results, and these do have some limitations on the number of variables used in a regression model).

Making the automatic variable selection work is done by creating a ***dynamic list*** of independent variables to feed into the model.

DYNAMIC LIST CREATION

After identifying which variables to include in the regression model, we had to find a way to make this list completely dynamic. We wanted to be able to specify the physician specialty code and hit "SUBMIT", without having to identify all of the variables to include in the model. Therefore, the program had to be intelligent enough to create its own list of independent variables.

To create a dynamic set of independent variables for our regression model, we first had to create a data set with all of the variable names in it. The value of the variable in our data set had to be a string which contained the actual variable name. We did this by concatenating the procedure code to a prefix. For example, "PROC3_784" was the name of a variable that described how much business a provider did for all procedures starting with the digits "784". See the code below:

```
data indvars;
```

```
    set filter;
    var = compress("proc3_"||proc3);
run;
```

Once all of the variable names were captured in a data set called “ALL_INDVARS”, we concatenated all of the names into one long string using the following code:

```
data temp;
    retain newvar;
    length newvar $3000;
    set all_indvars;
        by dummy;
    if first.dummy then do;
    if &depvarc = 'tot_amt' then newvar='num_hicns num_procs pct_female msa_flag
             pct_ge65 pct_atrisk_high ';
    if &depvarc ne 'tot_amt' then newvar='log_num_hicns log_num_procs pct_female
            msa_flag pct_ge65 pct_atrisk_high ';
    end;
    newvar = trim(newvar)||' '||compress(var);
    if last.dummy then call symput('indvars',newvar);
run;
```

Note that we initialized the string NEWVAR first with a set of core independent variables that were to be included in every regression model. Then, we added each successive variable VAR onto the NEWVAR variable using the TRIM, COMPRESS, and concatenation (||) functions. It is important to use the TRIM and COMPRESS functions to eliminate any leading and embedded spaces. Once completed, we assigned the string to a global macro variable called INDVARS. Also note the use of the FIRST. And LAST. Statements; we use them to identify the first and last observations of the data set. Sometimes it’s just easier to explicitly control when variable assignment occurs.

The INDVARS macro variable is resolved later in the PROC REG to perform the regression analysis:

```
proc reg data = without_outliers outest=estimates;
    predicted: model &depvar =  &indvars / selection = forward slentry=.05;
    title1 'PROC REG:  WITHOUT OUTLIERS';
    title2 "Dependent Variable = &depvar";
run;
```

OUTLIER REMOVAL

There are several methods of removing outliers from a regression model; we employed three different methods in our analysis. Since we were concerned with automating this process, we created a macro variable that allowed the user to quickly choose their outlier removal method. We focused on the following three approaches:

1) Removing observations with a residual z-score >= 2.5 or <= -2.5

2) Removing observations with an amount z-score >=2.5 or <= 2.5

3) Removing observations if their amount is either a) greater than the 99th percentile or b) their number of patients is greater than the 99th percentile

With method 1, a PROC GLM was run first with all observations, and residuals and their accompanying z-scores were calculated. Those providers with a residual z-score above 2.5 or less than -2.5 were pulled from the data set before running the PROC REG, and then put back in when running the PROC SCORE.

To implement the outlier removal method, at the top of the SAS program, the user had to type 1, 2, or 3 in the %let method = statement to identify which outlier removal method they would like to use.

Having a quick way to change the algorithm was very useful in comparing the results of various regression runs. When I run regressions, I do so in an iterative fashion, comparing results along the way with previous

models. Having this macro variable made running the various models much more efficient than blocking out code to try different outlier removal methods.

MODEL DEVELOPMENT

After removing the outliers from the final data set, we employed PROC REG to run the regression model. We used our macro variables DEPVAR and INDVARS to define our dependent and independent variables. DEPVAR in most cases was the log (base 10) of the total amount a provider billed to Medicare procedure codes. In other models run, we assigned the raw amount to the macro variable DEPVAR for comparison purposes. The code we submitted is shown below:

```
proc reg data = without_outliers outest=estimates;
    predicted: model &depvar =  &indvars / selection = forward slentry=.05;
    title1 'PROC REG:  WITHOUT OUTLIERS';
    title2 "Dependent Variable = &depvar";
run;
```

We used the SELECTION = FORWARD options with the SLENTRY option set to .05, meaning a variable was only kept in the model if it contributed significantly to the model with a p-value of less than or equal to .05. The estimates are then output to a data set called “ESTIMATES”, which is later used in the PROC SCORE.

Then, to run the model again with ALL of the data (outliers included), we had to use PROC SCORE with the TYPE= options set to PARMS. The TYPE = option tells SAS that the values in the ESTIMATES data set are to be used as estimates for the variables.

```
proc score data=final score=estimates out=results type=parms predict;
    var &depvar &indvars;
run;
```

CLUSTER ANALYSIS

For some specialty codes, cluster analysis was warranted because the providers were actually sub-grouped according to place of service or particular procedure codes. Clustering helps ensure that what may be an outlier in one group is not necessarily an outlier for another group. In our analysis, we decided to run a cluster analysis on psychiatrists because they generally fell into two groups according to place of service: office or non-office. Therefore we set our macro variable CLUSVARS to POS11, the variable representing the percentage of business done in an office setting. See the code below:

```
%let clusvars = pos11;

proc cluster data=temp outtree=tree method=average p=100 ccc pseudo noprint;
    id upin_id;
    var &clusvars;
run;

proc tree data=tree out=clusters n=3 noprint;
    id upin_id;
    copy &clusvars;
run;
```

Running the above code yielded three strong clusters:

Cluster #	N=	Mean Allowed Amount	% Business done in an Office Setting
1	93	$24,774	10.8%
2	194	$8,216	98.8%
3	25	$23,997	61.9%

The cluster analysis reveals that the majority of psychiatrists see their patients in an office setting (cluster 2), while a sub-group sees them elsewhere (cluster 1). Cluster 3 is somewhere in the middle. We then merged the cluster data set back in with the provider data before running our models. Then when running our

models, we added a "BY CLUSTER" statement in the PROC REG and PROC SCORE. This approach yielded separate results for each cluster and helped to more accurately define outliers.

FLAGGING THE OUTLIERS

After running the PROC REG and PROC SCORE, we calculated the residuals for each provider. The residual is the actual total allowed amount minus the predicted allowed amount as calculated by our model. Then, we calculated a z-score for each residual, using the following SAS code:

```
proc stdize data=results out=results2 method=std pstat;
    title2 'STANDARDIZE THE RESIDUAL TO Z-SCORE';
    var residual_std;
run;
```

Another way to do this is to calculate the z-score manually:

z-score = (residual – residual mean) / residual standard deviation

The z-score has a mean of 0 and a standard deviation of 1 and is frequently used to flag outliers.

Once the z-scores were calculated, we identified our potential fraudulent providers by flagging those with a residual z-score greater than or equal to 2.5 AND a total allowed amount greater than or equal to $10,000.

MODEL COMPARISON

When conducting regression analysis, you obviously want the best-fitting model. But how do you know when you've got the best model? This is perhaps the most difficult part of regression analysis: when to call it quits. We used three methods in determining the "goodness of fit" of our models:

1) *r-square value:* PROC REG produces an r-squared value between 0 and 1, 1 indicating a perfect fit and 0, no fit. Generally, the closer our r-squared value was to 1, the better.

2) *residual standard deviation:* We also looked closely at our total residual standard deviation. In this case, the smaller, the better.

3) *number of flagged providers*: Reviewing the number of flagged providers also tells us how our model is doing in predicting allowed amounts. If 20% of the providers are flagged, we know our model is capturing too many providers. Generally, only a handful of observations should be flagged as true outliers, perhaps 5 percent.

CONCLUSION

When you have to run a regression model many times, automating some of the processes can save a lot of time. In our Medicare fraud-detection example, the biggest time-saver was creating a dynamic list of variables for each provider specialty code. This step allowed us to avoid tedious hard-coding of all of the procedure, diagnosis, and place-of-service codes as variables. Additionally, you don't necessarily need SAS Enterprise Miner to have a "point-and-click" approach to predictive analysis. Although Enterprise Miner is a very effective tool, you can accomplish many of the same tasks using Base SAS and some procedures from SAS/Stat.

REFERENCES

The National Health Care Anti-Fraud Association (NHCAA) Home Page. *"Health Care Fraud: A Serious and Costly Reality For All Americans".* < http://www.nhcaa.org/pdf/all_about_hcf.pdf>

Reprinted with permission of SAS Institute Inc. from SAS Technical Support documentation. *"Convert missing values to zero and values of zero to missing for numeric variables".* Copyright 2004 SAS Institute Inc. <http://ftp.sas.com/techsup/download/sample/datastep/numeric.html>, Cary, NC: SAS Institute Inc.

Centers for Medicare and Medicaid Services website. HCPCS file, 2005, downloaded from the link: <http://www.cms.hhs.gov/providers/pufdownload/anhcpcdl.asp>

ACKNOWLEDGMENTS

The author would like to thank the following people for their assistance in writing this paper:

Helen Paine, SRA International

Venkat Chalasani, SRA International

Bob Bandzwolek, SRA International

Dave Vennergrund, SRA International

David Schwartz, SRA International

Tri-Centurion staff (www.tricenturion.com)

CONTACT INFORMATION

The author may be contacted at:

Christy Warner
SRA International
4300 Fair Lakes Court
Fairfax, Virginia 22033
703-803-1863 (phone)
703-502-7761 (fax)
christy_warner@sra.com
www.sra.com