

A SAS® Macro Solution for Scoring Test Data with Output Features and Parameters from Logistic Regression Model Training

Joseph Blue, ID Analytics, San Diego, CA

Abstract

A traditional technique in building predictive models involves isolating raw data into training and testing sets. The standard procedure includes building the model with the training data then evaluating its performance on the previously-unseen “test” data. For situations in which the dependent variable is binary, Logistic Regression is a common statistical approach for associating a probability with the response. PROC LOGISTIC offers several variable-selection methods which can result in an optimally-trained model with a subset of the original input variables. However, if the input variable list is large (potentially up to thousands of variables), calculating the model score for the test data becomes protracted. This approach offers an automated SAS® Macro solution for scoring a testing set using the model features and parameters selected through logistic regression.

Introduction

Logistic regression is one of several useful data mining procedures which can be used to estimate the probability of a binary response by detecting differing patterns in the independent variables between the response categories. With a representative sample and the right variables selected (features) the user can assemble a model which will identify the response a relatively large percent of the time while keeping high-scoring non-responses (i.e. false positives) to a minimum.

PROC LOGISTIC offers several methods for variable selection which may identify a different subset of the independent variable list and their coefficients. Depending on the model performance, the modeler might want to modify the input variables or create new features. Any additions to the input variables or method of variable selection could produce a different set of features and weights.

The results of the model training set gives an impression about how well the model is functioning, but for an accurate, “real life” assessment of the model performance, the modeler would like to know how the data reacts to observations it has not encountered before (i.e. – data that was not included in the training process). This is the impetus for creating the “test” set. To score the test set, the user needs to duplicate the logistic equation determined during training.

What follows is an automated process for calculating the probability of response for the test set which would reduce time spent editing the parameters of the logistic equation and eliminate the possibility of transcription or rounding errors.

Process Overview

Run PROC LOGISTIC on training data

Train the model using PROC LOGISTIC. The goal of this step is to produce the dataset *param*, which contains the model parameter estimates.

*** Important – the *outest* option must be used to capture the coefficients of the logistic response function.

```
proc logistic
  DATA = train
  descending
  outest = param ;
  model tag= <ind variable list>
  / SELECTION = STEPWISE
  /* or BACKWARD, FORWARD, etc */
  CTABLE;
run;
```

Use SAS® to score the test data?

Scoring the test data with the parameters obtained in training can be done by PROC LOGISTIC...almost. The problem is that variables

not selected in training will have a value of *missing*, which will cause an error at run time, because SAS expects a value for all parameters in the dataset specified in the *inest* option. An iterative loop through the variables to change missing parameter values to 0 would be one way to circumvent this setback. Also required is another execution of the LOGISTIC procedure to produce the scores, which might not be desirable in the case where an incredibly large dataset is involved. Furthermore, if SAS/STAT is not available to the user, then this method would not be compatible for weights obtained from historical analyses or generated from another data mining procedure.

```
proc logistic
  DATA = train
  descending
  inest = param ;
  model tag= <variable list>
    / maxiter=0;
  output out=test2 p=p;
run;
```

An alternative method which requires only one call to PROC LOGISTIC is presented below.

SAS® Macro to build an array of Model Parameters

```
%macro makearray(dset);
```

*** Input dataset is output from PROC LOGISTIC. In this macro, PROC CONTENTS is run to get the variable list. If the variable does not begin with “_” (internal SAS variables), then &testparms.x is populated with the variable name, where x ranges from 1 to the number of variables (&numparms);

```
proc contents
  data=&dset noprint
  out=cont_out;
proc sort data=cont_out; by npos;
```

```
data cont_out;
  set cont_out end=last;
  if _n_=1 then count=0;
  if substr(name,1,1)^='_'
    then count+1;
  if substr(name,1,1)^='_'
    then call symput
      ("testparms" || trim
        (left (count)), trim
        (left(name)));
  if last then do;
    call symput
      ('numparms',trim
        (left(count)));
  end;
```

```
run;
```

* In this step, the macro iterates through the “pseudo array” *testparms* and only those variables with a parameter value present (i.e. – features selected by the model) are selected. When completed, this step will have created another pseudo array *parms* which will have %eval(&goodparms) elements ;

```
data _null_;
  set &dset(firstobs=1 obs=1);
  count=0;
  %do i=1 %to &numparms;
    if &&testparms&i^=.
      then do;
        count+1;
        call symput
          ("parms" || trim
            (left(count)),
            trim(left
              (&&testparms&i)));
      end;
  %end;
  call symput
    ('goodparms',
    trim(left(count)));
```

```
run;
```

```
%mend;
```

```
%macro printarray;
```

* This short macro simply creates a standard array (“parms”) containing the variable names that the model has selected. This will be used to generate the linear combination $\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$.

```
array parms{*}
  %do i=1 %to &goodparms;
    &&parms&i
  %end;;
```

```
%mend;
```

Score test data

* Declare &goodparms and &parms.x as global variables because they will be needed outside the **makearray** macro. The range of the loop is set to be the number of variables in the *param* dset (this number is an overestimate of the number of independent variables since there are internal SAS variables in *param*).

```
%let dsid = %sysfunc(open(&dset));
%let nvars = %sysfunc(atrn(&dsid,NVARS));
%let rc = %sysfunc(close(&dsid));
```

```
%global goodparms;
```

```

%do i=1 %to %eval(&nvars);
  %global parms&i;
%end;

%makearray(param);

* Associate the value of the coefficient with each
parameter that the logistic model selected:
&parvalue.y, where y ranges from 1 to
&goodparms ;

data _null_; set param ;
  %printarray;
  do i=1 to dim(parms);
    call symput
      ('parvalue' || trim(left(i)),
       trim(left(parms{i})));
  end;
run;

```

* Calculate the linear combination of the coefficients of the model developed in training multiplied by the value of the variable in the test set. If an intercept is used, it will be &parvalue1 and the coefficient =1.0. If no intercept is requested, then the equation to calculate score will have to be modified. The estimated probability is calculated from the logistic response function (1).

$$E(p) = [1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}]^{-1} \quad (1)$$

```

data test; set test;
  %printarray;
  score = &parvalue1
    %do j=2 %to &goodparms;
      + &&parvalue&j*parms{&j}
    %end;;
  p = 1.0 / (1.0 + exp(-score));
run;

```

The dataset *test* is now ready to be evaluated by various measures to determine effectiveness of the model on predicting the response.

If more variables are added to the logistic model statement, or another variable selection method is requested, then it is likely a new set of features and parameters will be selected. When added to the training process, this program will insure that the new solution set is being used on the test set to estimate the probability without requiring the user to make any changes to the evaluation portion of the code.

Example

The data used to illustrate this program was a sample of census data obtained from IPUMS (Integrated Public Use Microdata Series) containing over 20,000 observations. The formal citation and URL address are located in the Acknowledgements section. This data was not chosen to study a specific research question, but was used to illustrate a specific instance where the program would be effective.

For this example, the probability of an individual being a veteran was modeled using variables in the following categories: Housing Status, Family, Race, Education, and Income.

Sampling & Training the Model

```

**** sampling dataset ****;
data train test; set in.ipums;
  *** note: some variable
  *** manipulation removed ****;
x=ranuni(26714690);
if x<0.5 then output train;
else output test;

```

```

**** training model with
**** backward selection ****;

```

```

proc logistic
  data=train
  descending
  outest=param;
model isVet = isfarm mortgage
  value acrehous isCondo costelec
  costgas costwatr costfuel
  ftotinc rooms builtyr
  unitsstr famsize eldch yngch
  age isMale isMarried isAsian
  isBlack isWhite isCollege sei
  poverty
  /selection = backward ctable;

```

Features selected – dataset *param*:

Feature	Estimate	Pr > ChiSq
Intercept	-6.7168	<.0001
isfarm	1.1930	0.0003
value	-5.69E-07	<.0001
isCondo	-0.4952	0.0157
famsize	-0.1403	<.0001
age	0.0519	<.0001
isMale	2.9332	<.0001
isMarried	0.6916	<.0001
isAsian	-0.4921	0.0083
isBlack	0.3525	0.0037

isCollege	0.4952	<.0001
-----------	--------	--------

Scoring the Test Data

In this step, the macro should identify the model variables and obtain the correct parameter values.

```
%inc "./log_array.mac";
options symbolgen;

%macro read(dset);

%let dsid = %sysfunc(open(&dset));
%let nvars = %sysfunc(attrn(&dsid,NVARS));
%let rc = %sysfunc(close(&dsid));
```

```
>>SYMBOLGEN: NVARS resolves to 31
>>This is the maximum number of parameters which
could be selected in training (if every variable were
determined to be significantly contributing to model
performance)..
```

```
%global goodparms;
%do i=1 %to %eval(&nvars);
    %global parms&i;
%end;
%makearray(param);
```

```
>>SYMBOLGEN: Numparms resolves to 26
>>This is the number of variables in the param dataset
that are not system-created (i.e. - do not begin with '_')
```

```
>>SYMBOLGEN: TESTPARMS1 resolves to Intercept
>>SYMBOLGEN: TESTPARMS2 resolves to isfarm
>>SYMBOLGEN: TESTPARMS3 resolves to mortgage
...<several lines deleted>
>>SYMBOLGEN: TESTPARMS25 resolves to sei
>>SYMBOLGEN: TESTPARMS26 resolves to poverty
>> These are all of the dependent variables in the
model.
```

```
>>SYMBOLGEN: GOODPARMS resolves to 11
>> This is the number of parameters which are non-
missing in the param dataset (i.e. variables selected in
training)
```

```
>>SYMBOLGEN: PARMS1 resolves to Intercept
>>SYMBOLGEN: PARMS2 resolves to isfarm
>>SYMBOLGEN: PARMS3 resolves to value
>>SYMBOLGEN: PARMS4 resolves to isCondo
>>SYMBOLGEN: PARMS5 resolves to famsize
>>SYMBOLGEN: PARMS6 resolves to age
>>SYMBOLGEN: PARMS7 resolves to isMale
>>SYMBOLGEN: PARMS8 resolves to isMarried
>>SYMBOLGEN: PARMS9 resolves to isAsian
>>SYMBOLGEN: PARMS10 resolves to isBlack
>>SYMBOLGEN: PARMS11 resolves to isCollege
>> These are the features ( $\beta$ 's) of the logistic function.
```

```
**** associate values of
**** parameters with parameter
**** names ****;
```

```
data _null_; set &dset ;
```

```
%printarray;
do i=1 to dim(parms);
    call symput
        ('parvalue' ||
        trim(left(i)),
        trim(left(parms{i})));
end;
run;
```

```
>>SYMBOLGEN: PARVALUE1 resolves to -6.716760138
>>SYMBOLGEN: PARVALUE2 resolves to 1.1929944534
>>SYMBOLGEN: PARVALUE3 resolves to -5.692591E-7
>>SYMBOLGEN: PARVALUE4 resolves to -0.495246084
>>SYMBOLGEN: PARVALUE5 resolves to -0.140291444
>>SYMBOLGEN: PARVALUE6 resolves to 0.0518505552
>>SYMBOLGEN: PARVALUE7 resolves to 2.933192778
>>SYMBOLGEN: PARVALUE8 resolves to 0.6915793357
>>SYMBOLGEN: PARVALUE9 resolves to -0.49206293
>>SYMBOLGEN: PARVALUE10 resolves to 0.3524726399
>>SYMBOLGEN: PARVALUE11 resolves to 0.4951709403
>>These values should match the parameter estimates
contained in the table in the model training section.
```

```
**** associate values of
**** parameters with parameter
**** names ****;
```

```
data test; set test;
    %printarray;
    score = &parvalue1
        + &&parvalue&j*parms{&j}
    %end;;
    p = 1.0 / (exp(-score) + 1.0);
run;
```

```
%mend read;
```

```
%read(param)
```

The probability of being a veteran (p) has now been estimated for the test cases using the parameters obtained during training. If the selection method were changed or more features were added, the parameters would most likely change and the test cases would need to be rescored. Modifying the values of the equation after each training is almost trivial for the relatively small number of features. However, even in a small example such as this, by automating the process, we can skip the editing step and remove the possibility of transcription or rounding errors every time the model is re-trained.

Evaluating Performance

Of interest to the modeler might be the score cutoff-point (i.e. – threshold) which would maximize the percent of positive responses while keeping the ratio of non-responses (false positives) within a limit that is acceptable for the given cost associated with making an incorrect prediction.

The following table could be used to assess the performance of the model on the test data:

Score Threshold	Percent of All Veterans above Cutoff	False Positives : Response Ratio
0.85	0.44	0.40 : 1
0.80	1.32	0.73 : 1
0.75	3.69	0.62 : 1
0.70	6.77	0.71 : 1
0.65	12.83	0.57 : 1
0.60	19.60	0.52 : 1
0.55	26.98	0.52 : 1
0.50	34.27	0.57 : 1
0.45	42.00	0.62 : 1
0.40	50.09	0.67 : 1
0.35	57.21	0.74 : 1
0.30	64.50	0.86 : 1
...
0.05	89.89	3.04 : 1
0.00	100.0	8.59 : 1

The table shows that considering a subjects with a score at or above 0.40 would correctly identify over 50% of veterans while misclassifying 0.67 non-veterans for ever veteran.

Conclusions

Transcribing model features and parameter values each time a model is trained can be the source of typographical and rounding errors. As the number of potential features in the model grows, the need for automation of calculating the probability for the test data becomes more evident. In utilizing such a procedure, the modeler can bypass a time-consuming step and potential source of errors, leaving more time to interpret the results and consider the addition of features that may improve the model's ability to estimate the probability of a response more efficiently.

Acknowledgements

The census data used as an example originated from the following source:

Steven Ruggles and Matthew Sobek et. al.
Integrated Public Use Microdata Series: Version 2.0
Minneapolis: Historical Census Projects,
University of Minnesota, 1997

Similar samples can be downloaded at the following URL:
<http://www.ipums.org>

Contact Info

Comments or suggestions can be directed here:

Joseph Blue
Senior Scientist
ID Analytics
jblue@idanalytics.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.