Paper 1334-2017

Predictive Models: Storing, Scoring and Evaluating

Matthew Duchnowski, Educational Testing Service

ABSTRACT

Predictive modeling may just be the most thrilling aspect of data science. Who among us can deny the allure of observing a naturally-occurring phenomenon, conjuring a mathematical model to explain it and then using that model to make predictions about the future? Though many SAS® users are familiar with using a data set to generate a model, they may not utilize the awesome power of SAS to "store" their model and "score" other datasets. In this paper we will distinguish between parametric and non-parametric models and discuss the tools that SAS provides for storing each and using them to score a cross-validation set. We will end with a brief survey of common measures often used for evaluating models.

INTRODUCTION

In the context of this paper, predictive modeling will involve splitting a dataset into two parts: the model building set (MB) and an independent cross-validation set (XV). Models are built using the MB set and "stored" so that they can later be applied to XV. The term "scoring" will be used to describe the process of applying a model to the XV dataset and generating a prediction for each observation. The XV dataset will then contain not only a *predicted* value but also a *true* value, making it the perfect tool for evaluating the efficacy of the model.

SPLITTING SAMPLES

If a data scientist is fortunate, he or she has access to a large and robust dataset. Once the data are procured, the SURVEYSELECT procedure is the perfect tool for randomly splitting the full dataset into smaller, usable subsets. The procedure can be used to sample according to complex weighting schemes and stratification methods or to draw a simple random sample. For example, we can select 100 random observations as follows:

```
proc surveyselect data=sashelp.heart out=MB n=100 outall;
run;
```

The `outall` option is used to retain all observations from the data source and place them in the output dataset, adding a single numeric variable, `SELECTED`. The variable `SELECTED` equals 1 for those observations in the chosen sample and 0 otherwise.

We are now in a position to create a cross-validation set by sampling records from the unselected portion:

```
proc surveyselect data=MB out=XV (drop=SELECTED) n=100;
      where SELECTED=0;
run;
```

We then drop the excessive records from MB:

```
data MB (drop=SELECTED);
      set MB;
      where SELECTED =1;
run;
```

We now have two randomly-equivalent and disjoint datasets containing 100 observations each. The data has many numeric and character variables, all taken from the SASHELP.HEART dataset (Framingham Heart Study). **Table 1** lists those variables that will be used throughout this paper:

Variable Name	Description	Variable Type
BP_Status	Blood Pressure Status ('Optimal', 'Normal', 'High')	Character
Cholesterol	Cholesterol	Numeric
Height	Height (inches)	Numeric
Sex	Sex of patient ('Male', 'Female')	Character
Weight	Weight (lbs)	Numeric

Table 1. Variables of SASHELP.HEART used in this paper

PARAMETRIC MODELS

Parametric models express a set of quantities as explicit functions of so-called *independent* variables. The model may have a form that is often commonly referenced by a name (eg. "linear regression"). The model can be reconstructed by anyone possessing the model form and parameter set. A *dependent* variable can then be predicted using the model and data.

Building and storing parametric models in SAS involves saving the resulting parameter estimates produced by one of many model building procedures.

Parameters can be stored two basic ways in SAS:

1. A *dataset* is generated with variables that have key names that can later be identified by the SCORE procedure.

2. An *item store* is created by the model building procedure that can later be referenced by the PLM procedure.

These two approaches will be illustrated in combination with various ways obtain predicted values.

THE OUTEST= OPTION

Parameter estimates can be exported to a dataset using the `outest` option as shown in the following regression:

```
proc reg data=MB outest=regModel;
      P_Cholesterol : model Cholesterol = Weight Height;
run; quit;
```

In this example, parameter estimates have been written out to the dataset `regModel` as seen here :

MODEL	_TYPE_	_DEPVAR_	_RMSE_	Intercept	Weight	Height	Cholesterol
P_Cholesterol	PARMS	Cholesterol	47.4506	458.8313	0.5339	-4.6373	-1

By using a colon in the model statement, we have assigned the name `P_Cholesterol` to the model. The name is significant not only in identifying the model parameters stored within `regModel` but also in identifying the variable that will contain predicted values on XV once scoring has occurred. That scoring process is carried out by **`proc score`** as shown below:

```
proc score data=XV score=regModel type=parms predict out=XV;
     var Weight Height;
run;
```

The scored dataset is written out to a new dataset using the `out=` option. The dataset now has variable `P_Cholesterol` that contains predictions for all observations as dictated by the `P_Cholesterol` model. In our example above, we have chosen to write the XV dataset over itself. If multiple models are attempted with alternate specifications, the user may find it beneficial to create a unique dataset for each model applied and leave the original XV unaltered. This approach would then involve managing those scored datasets. In these examples however, we will continue to write the results directly to the XV set.

Note that some parametric modeling procedures support multiple model builds within a single execution:

```
proc reg data=MB outest= regModel;
     X : model Cholesterol = Weight;
     Y : model Cholesterol = Height;
     Z : model Cholesterol = Weight Height;
run; quit;
```

In this case, the dataset `regModel` will contain parameter estimates for each of the models X, Y and Z. All models captured in the dataset can be applied to XV using a single **`proc score`** as above. Note that the models must be uniquely named so that there is no variable naming conflict for the predicted values on XV.

THE OUTMODEL= OPTION

Similar to the `outest=` option, the `outmodel=` option is used to write parameter estimates to a dataset. In this example, we use PROC LOGISTIC to model the likelihood that character variable `Sex` is 'Male':

```
proc logistic data=MB outmodel=logitModel;
     model Sex(Event='Male') = Weight Height;
run;
```

The model is then applied to XV using a `score` statement supported by **`proc logistic`**:

```
proc logistic inmodel=logitModel;
     score data=XV out=XV;
run;
```

On the XV dataset, true values of Sex are copied into a new variable F_Sex (From: Sex) and model predictions are directed to a new variable I_Sex (Into: Sex).

If the model does not need to be stored, the user has the option of combining the two procedures above by dropping the `outmodel/inmodel` options and writing a single procedure. For example, a cumulative logit model is built using MB and applied to XV in a single procedure below:

```
proc logistic data=MB;
      model BP_Status = Weight*Height/ link=CumLogit;
      score data=XV out=XV;
run;
```

THE STORE STATEMENT

A growing number of model building procedures in SAS support the `store` statement. This statement differs from `outest=` and `outmodel=` in that it creates a SAS library member known as an *item store*. The item store is held in memory and can be accessed by the `restore=` option in the *post-linear modeling* procedure PROC PLM.

Below we see the creation of an item store:

```
proc orthoreg data=MB;
      class Sex;
      model Cholesterol = Sex | Height | Weight;
      store orthoModel;
run;
```

The item store is referenced by a `restore=` statement as follow:

```
proc plm restore=orthoModel;
      score data=XV out=XV pred=P_Cholesterol;
run;
```

The predictions are stored in XV under variable `P_Cholesterol`, as dictated by the `pred=` option.

THE CODE STATEMENT

There is one additional way to store a parametric model which bears mentioning, though it is less flexible than those methods mentioned previously: the `code` statement. Some model building procedures can use this statement to store the resulting model in the form of an algorithm coded in SAS syntax. To apply this method, the syntax must be directed to an external file location where the user has write-access. An example is shown below:

```
proc glm data=MB noprint;
      class Sex;
      model Cholesterol = Height | Weight;
      code file='C:\glmModel.sas';
quit;
```

The scoring algorithm can then be accessed by an `%include` statement executed within a data step:

```
data XV;
      set XV;
      %include 'C:\glmScore.sas';
run;
```

An additional variable `P_`*varname* is added to XV, where *varname* is the name of the model's dependent variable.

NON-PARAMETRIC MODELS

Unlike parametric models, so-called "non-parametric" models cannot be conveniently codified using a dataset or item store. Nevertheless, there are a set of non-parametric model building procedures that allow the user to apply the resulting model to an independent cross-validation set.

THE SCORE STATEMENT

Within select procedures, the `score` statement can be used to score XV while constructing a model based on MB, all within a single procedure. Below, we see that the model building and cross-validation processes both occur within a single procedure.

```
proc tpspline data=MB;
   model Cholesterol = (Height Weight);
   score data=XV out=XV;
run;
```

Again we have overwritten XV with itself. It has a new variable `P_Cholesterol` that contains a prediction for each observation as dictated by the `P_Cholesterol` model. An additional variable is added, `P_`*varname*, where *varname* is the name of the dependent variable used in the model statement.

Many parametric modeling procedures also offer this `score` option, as noted previously. In this instance however, the `model` and `score` statements must appear together in a single procedure.

SUMMARY

A list of popular modeling building procedures is listed in **Table 2** along with information about whether the procedure supports `outest=` or `outmodel=` options and/or the `store` or `code` statements. The final column indicates if the model building procedure supports its own `score` statement that can be used to score a cross-validation set. This is not to be confused with the `score` statement used by **`proc plm`** to apply models that are stored using `store`.

SAS PROCEDURE	OUTEST=	OUTMODEL=	STORE	CODE	SCORE
PROC ADAPTIVEREG					✓
PROC CALIS / TCALIS	✓	✓			
PROC GAM					✓
PROC GENMOD			✓	✓	
PROC GLIMMIX	✓		✓	✓	
PROC GLM			✓	✓	
PROC LIFEREG	✓		✓		
PROC LOESS					✓
PROC LOGISTIC		✓	✓	✓	✓
PROC MIXED			✓	✓	
PROC NLIN	✓				
PROC ORTHOREG	✓		✓		
PROC PHREG	✓		✓		
PROC PROBIT	✓		✓		
PROC REG	✓		✓		
PROC SURVEYLOGISTIC			✓		
PROC SURVEYPHREG			✓		
PROC SURVEYREG			✓		
PROC TPSPLINE					✓

Table 2. Sample model-building procedures that employ OUTEST=/OUTMODEL=, STORE, CODE and SCORE

EVALUATING

Most model-building procedures in SAS will generate, by default, some kind of statistic suitable for evaluating model quality. However, because our goal here is to generate an independent evaluation on XV, some additional processing will be needed.

Since there is no single metric for evaluating predictive models, we have chosen a small collection of favorites to describe here.

R-SQUARE

The *R-Square* measure describes the proportion of the dependent variable's variance that has been explained by the model. Linear Regression modelling procedures will almost always produce this measure by default for the model building data set. The code below will calculate R-Square for predictions on XV:

```
%let true = Cholesterol;
%let pred = P_Cholesterol;
%let dataset = XV;

proc sql;
      select 1-_SSE_/(_SSE_+_SSR_) as _RSQUARE_
      from(
            select
                  SUM((A.&pred.-A.&true.)**2) as _SSE_ ,
                  SUM((A.&pred.-B.Ybar)**2) as _SSR_ ,
                  SUM((A.&true.-B.Ybar)**2) as _SST_
            from
                  &dataset. as A,
                  (select MEAN(&true.) as Ybar from &dataset.) as B
      ) ;
quit;
```

MSE AND RMSE

Mean Square Error (MSE) and *Root Mean Square Error* (RMSE) are both popular ways for evaluating and comparing model accuracy. Each can be computed directly:

```
%let true = Cholesterol;
%let pred = P_Cholesterol;
%let dataset = XV;

proc sql;
      select
            mean((&pred.-&true.)**2) as _MSE_,
            sqrt(mean((&pred.-&true.)**2)) as _RMSE_
      from &dataset.;
quit;
```

QUADRATIC-WEIGHTED KAPPA

When a dependent variable is ordinal and categorical, it is good practice to review a cross-tabulation of the predicted categorization versus the true categorical values. Cohen's *kappa statistic* is a popular way of quantifying the results of this cross-tabulation when the number of categories is larger than 2.

Let us suppose a cumulative logistic regression has been carried out for dependent variable `BP_STATUS`, which takes on the values 'High', 'Normal' and 'Optimal'.

The *quadratic-weighted kappa* can be generated using PROC FREQ as shown below.

```
proc freq data=XV order=internal;
      tables F_BP_Status*I_BP_Status / agree (WT=FC);
      test kappa wtkap;
run;
```

SAS assumes category order matches the alphabetical ordering of internal values. In this example it so happens that alphabetical order or internal values matches the appropriate category order and so `order=internal` is specified (which is also the default). If this is not the case, the user should position the variables in the data according to their order and `order=data` can be used.

The *linear-weighted kappa* and *simple kappa* statistics are also available through PROC FREQ.

MISCLASSIFCATION RATE

In the case of logistic regression, a model classifies each observation based on a probability. The model's ability to classify observations accurately can be measured using the *misclassification rate*. This value is the total number of false-positive and false-negative classifications divided by the total number of classifications made. The measure can be output using the `fitstat` option as seen below:

```
proc logistic inmodel=clogitModel;
      score data=XV out=XV fitstat;
run;
```

The misclassification rate is produced under the heading **ERROR RATE**.

.

AREA UNDER ROC CURVE

The ROC (Receiver Operating Characteristic) curve is a popular diagnostic tool and the *area under the ROC curve* (AUC) is commonly used to assess binary response models such as logistic models. If the AUC statistic is all that the user needs, then the `fitstat` option can be used (provided that the data has binary responses). If the user wants to see the actual ROC curve, it is required that `ods graphics` is turned on and that the `model` and `score` statements are used together without an `inmodel=` option.

```
ods graphics on;
proc logistic data=MB rocoptions(id=cutpoint);
      model Sex(Event='Male') = Weight Height;
      score data=XV out=XV outroc=rocdata;
run;
ods graphics off;
```

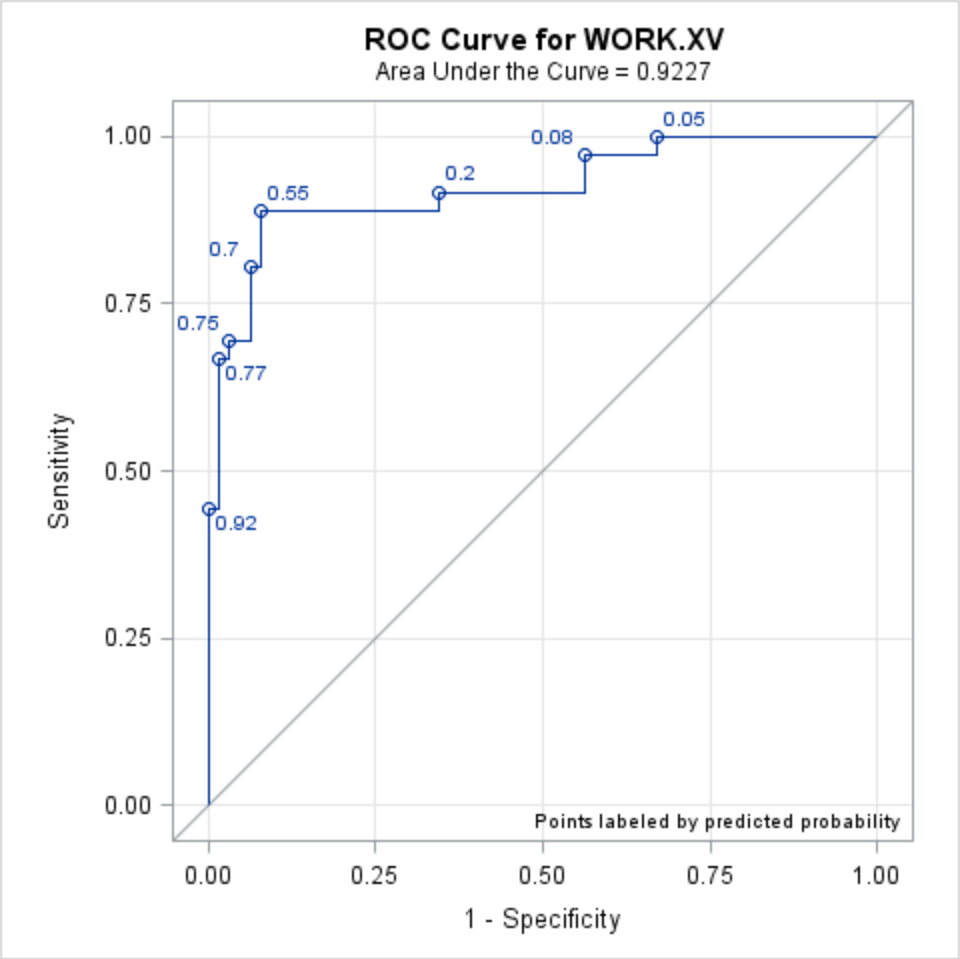

Figure 1. Output for ROC Curve on XV data

Note that the area under the curve is printed in the graph title. In the **`proc logistic`** header, the `plots` option has not been specified. If it were, then the ROC curve would be drawn for the MB data (by default). To produce the curve for XV, we use the `outroc=` option with our score statement. Not only does this trigger the creation of our ROC curve for the dataset being scored (XV), but it also produces an output dataset `rocdata` that contains information necessary for drawing the ROC.

CONCLUSION

With very little code, it is possible for even beginner-level data scientists to start storing their models and scoring cross-validation sets. Though the methods may differ slightly between model building procedures, each allows the models to be stored for subsequent application. The power to store and evaluate models opens up possibilities for collaborative efforts and long-term statistical modeling projects.

REFERENCES

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational & Psychological Measurement, 20*(1), 37–46. https://doi.org/10.1177/001316446002000104

Duchnowski, M. (2014). Nonnegative least squares regression in SAS [Paper no. 1829-2014). In *Proceedings of the SAS Global Forum, Washington, DC.* Retrieved from http://support.sas.com/resources/papers/proceedings14/1829-2014.pdf

Fleiss, J. L.; Cohen, J., & Everitt, B. S. (1969). Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin, 72*, 323–327. https://doi.org/10.1037/h0028106

Gutierrez, D. D. (2015). *Machine learning and data science: An Introduction to statistical learning with applications in R.* Basking Ridge, NJ: Technics Publications.

Wicklin, R. (2014, February 17). The missing value trick for scoring a regression model [Blog post]. Retrieved from http://blogs.sas.com/content/iml/2014/02/17/the-missing-value-trick-for-scoring-a-regression-model.html

Wicklin, R. (2014, February 19). Techniques for scoring a regression model in SAS [Blog post]. Retrieved from http://www.statsblogs.com/2014/02/19/techniques-for-scoring-a-regression-model-in-sas/

SAS. (2015, July 15). *Usage note 39724: ROC analysis using validation data and cross validation*. Retrieved from http://support.sas.com/kb/39/724.html

ACKNOWLEDGMENTS

Special thanks to Amanda Cirillo and Paul Hilliard for their feedback on this paper. I would like to thank my wife Robin, who provides me with unending support in all that I do.

CONTACT INFORMATION

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

Matthew Duchnowski
Educational Testing Service
Rosedale Rd, MS 20T Princeton, NJ 08541
(609) 683-2939
mduchnowski@ets.org