**MGS 8040 Data Mining**

**Basic Data Analysis with SAS**

Dr. Satish Nargundkar

**Case:** We have the task of building a classification model in the financial services industry, specifically for predicting the riskiness of customers for auto loans. The classification modeling project that we are beginning should result in a formula that lets us score each potential customer, the score being an evaluation of risk based on information about the customer’s past behaviors and current condition. The sources of information are the application, credit history, and characteristics of the transaction itself.

**Data Source**

Two datasets are provided, called data1 and xtra1, each containing 14,042 records (the same 14,042 customers who received and auto loan from a lender). The first dataset has over 35 variables, and the second one has over 20. They were provided by the customer separately, and need to be merged. As discussed in the previous section on the initial client meeting, the dependent variable is the status of the existing customers, in this case a simple two valued variable with values GOOD and BAD. There is a variable in one of the datasets provided called GOOD, with a value of 1 for customers judged by the lender to be “Good”, and those with value 0 for customers judged to be “Bad”. The dataset also happens to have a variable called BAD, which is the mirror image of the former, and is not necessary.

**Analysis Summary**

Before beginning analysis with SAS, it is useful to have a plan of what needs to be done. Building a classification model in this case requires us to perform Discriminant Analysis or Logistic Regression, both variants of regression appropriate for a categorical dependent variable. The distinctions between the two will be discussed later. Preliminary analysis of data is needed prior to building any multivariate models, however. The typical approach to entire project is as follows:

1. **Data Preparation**: Aggregating data, merging files, creating any new variables, cleaning the data (cleaning can be an ongoing process, as you may discover problems after some analysis).
2. **Training and Validation sets**: Splitting the dataset into two parts – a training dataset to build the model on, and a validation dataset to test that the model works on something other than what it was built on.
3. **Modeling:** The analysis of data proceeds typically in three stages. All of this analysis is done only on the training sample! Remember to keep the validation sample for use after the model has been built.
	1. Univariate Analysis: This is where we look at frequency tables or histograms as well as descriptive statistics like the mean, median, standard deviation, to identify distributions of all the variables, both independent and dependent. This step is necessary to ensure that the variables satisfy any modeling assumptions, as well as to find errors in data, to aid in data cleaning.
	2. Bivariate Analysis: Here, we look at scatter plots or cross-tabulations (more appropriate here due to categorical dependent) for relationships between each independent variable and the dependent variable individually. It helps us to assess whether the variable by itself seems to have some relationship with the dependent, and if so, the nature of that relationship. The relationship may be linear or non-linear, positive or negative. This can also help us see if the relationship makes intuitive sense given our business knowledge. If not, either we have discovered something unusual, or our data are simply wrong!

Bivariate analysis is also the key to breaking down a continuous independent variable into categories, if needed. Often times a variable, say Age, is not related in a smooth linear fashion with the dependent. We may discover that some age groups (chunks of values) behave similarly to each other, while being different from other age groups. So it may make more sense to use such age “buckets” as the variable values rather than the original ages. In other words, we convert age into a categorical variable and then use dummy variables to code those categories.

* 1. Multivariate Model: After we are satisfied that the data are reasonable and we have the variables in the form we want, we can proceed to the regression or other modeling technique to build a model that looks at the independent variables together. There are different approaches one can take here too. The approach we will take is to start will all the variables in the model, and remove insignificant ones in small steps until the final model has only those variables that are significant and meaningful.
1. **Evaluation:** Once the model has been built on the training sample, we evaluate its performance on both the training and the validation samples. Ideally, the performance of the model (in this case, the accuracy of classification of customers as “Good” or “Bad”) should be the same on both samples. If the performance on the validation sample is significantly lower than on the training sample, it typically indicates over-fitting, and the model needs to be simplified.
2. **Implementation/Monitoring:** Once the model has been found satisfactory based on the validation sample, it can be implemented. However, the real performance can only be assessed after implementation, so it must be monitored and its performance over time recorded. In case of significant discrepancy between expected and actual performance of the model, it may have to be adjusted or rebuilt. This typically happens because of environmental changes in the business. Models are built using past data (all data represent the past!), and people’s demographics can change, so can behavior and a host of other factors. So the model that worked on past data may not work as well in the future. How would you handle this problem?

**SAS Basics**

**Virtual Software Access**

You can turn your machine into a virtual GSU computer to access certain software, including SAS.

Go to [vcl.gsu.edu](http://vcl.gsu.edu/)

Open a new reservation -

Enter password and ID and you are ready to roll!

If you access virtual machine, be sure to save your project folder on the network drive (H), or you will lose it.

**Preliminaries**

**Folder Structure on your Computer**

Before beginning, you must be aware of where your data and programs are located. Create a folder called PROJECT on your machine (say the D drive, so you have D:\PROJECT). Within Project, create subfolders called DATA, PROGRAMS, FORMATS, and OUTPUT.

**DATA**: This should contain your data files – SAS datasets

**PROGRAMS**: This is where you save all the SAS programs you write

**FORMATS**: This contains a file called formats.sc2, a compiled library of formats that you use to create frequency tables of variables with bins defined the way you want.

**OUTPUT**: Save program output files in this folder.

In addition, it may be useful to create a **LOG** folder and store all the logs from program runs. For some businesses, this is important.

**Starting to Work with SAS**

**SAS Windows:** You launch SAS and see a **Program Editor** Window, a **Log** Window, and an **Output** Window, which is hidden behind the other two, but can be seen by clicking on the Output tab at the bottom. It also pops up automatically when there is output.

**Start Program:** You **begin every session** by running the ***start.sas*** program from the program folder. The purpose of this program is to define *libnames*, or shortcut substitutes for long paths to folders.

options compress=yes; run;

libname **save** "d:\project\data";

libname **library** "d:\project\formats";

run;

quit;

Here the words SAVE and LIBRARY will refer to the DATA and FORMATS folders after this program is run. To run the program, you can use the following ways:

1. Click on Run/Submit
2. Right click and select “Submit all”
3. Click on the icon of the running man
4. Press function key F8 on your keyboard

Once the program is run, the **log** should show a successful run in **blue lettering**.

**Looking at your data**

**If anywhere in the following, you see an error message saying a certain format does not exist, see the section on “Format Errors – Removing Formats.”**

We have in the auto loan project two files, data1.sd2 and xtra1.sd2. First, you can look at some metadata about the files as follows:

PROC CONTENTS **data**=save.data1;

Run;

This will show you how many records, variables the dataset called data1 has, as well as an alphabetical list of the variables. Note that the word **save** points SAS to the data folder as defined in the start program.

Now try the following:

PROC CONTENTS **data**=save.data1 position;

Run;

Besides alphabetical, this will also give you the list of variables in the sequence in which they occur physically in the dataset.

To actually **see the data values in rows and columns**, type the following:

PROC PRINT data=save.data1;

Run;

This will output all the records and variables. Now try

PROC PRINT data=save.data1 (obs=10);

Run;

Next, restrict the variables too as follows:

PROC PRINT data=save.data1 (obs=10);

Var AGEAVG DWNPMT VAGE;

Run;

What does this next program do?

PROC PRINT data=save.data1 (obs=10);

Var AGEAVG--VAGE;

Run;

It will print 10 records for all the variables between **ageavg** and **vage**, including those two.

**Output to html, for reading into Word, Excel**

Any program that creates output can be surrounded by two statements that create an html file, as shown below:

ODS html file=“D:\project\output\tenrecs.html”;

PROC PRINT data=save.data1 (obs=10);

Var AGEAVG--VAGE;

Run;

ODS html close;

The ODS html statement at the top and bottom enclose the previous program that output 10 records for the 3 specified variables. This same output will now be stored as the file ***tenrecs.html.***

**Sort and Merge Datasets**

We wish to sort data from both datasets based on a common Key variable, like account number, then merge the two files based on that.

PROC SORT **data**=save.data1; by acctno; run;

PROC SORT **data**=save.xtra1; by acctno; run;

DATA save.data2;

MERGE save.data1 save.xtra1;

By acctno;

Run;

Quit;

Note that the DATA save.data2 statement creates a new dataset called Data2, which is then populated by the merged data from the two existing files.

Sometimes, the values of the Key variable, like ACCTNO, do not match in the two files to be merged. Assume we want to merge the files such that all the account numbers in the first file are included in the new dataset, whether or not they exist in the second file. Try this program:

DATA save.data2;

MERGE save.data1(IN=a) save.xtra1;

By acctno;

IF a;

Run;

Quit;

**Creating Training and Validation Datasets**

If we build a prediction model using your entire dataset, we have no way of testing to see if it would actually work on data that has not been used to build it. In practice, our model will only be used in the future on new data. So we need some assurance that it will work on more than just the data that were used to build it. We validate the model by splitting our dataset into two parts, the training sample to build the model, and the validation sample to test it. We need enough data to build the model, so if the dataset is small, we may perhaps use as much as 80% of it to build and leave 20% for testing. If the dataset is large enough, as most modern datasets are, then split the data into 50% for training and 50% for validation.

DATA save.train save.valid;

SET save.data2;

*Random1* = RANUNI(14380132);

IF *Random1* < 0.5 THEN **output** save.train;

 ELSE **output** save.valid;

Run;

We are creating a new variable called *Random1*, and splitting the dataset into two based on the values of that variable. Note that RANUNI creates random numbers that are uniformly distributed, between 0 and 1. The function needs a seed number, which you make up. I made up a random seed number of 14380132. As long as the seed is the same, the random numbers generated are the same. This is useful if you delete the output files TRAIN and VALID after the project is done. If you have saved the program, you can recreate the same TRAIN and VALID files again as long you use the same seed number.

**Univariate Analysis**

We can look at frequencies of variables as a starting point of our analysis. Remember that from here on, all analysis should only be on the TRAIN dataset, until the model is built.

PROC FREQ data=save.train;

TABLES dwnpmt;

Run;

This will output the frequency table for the variable *dwnpmt*, the down payment by each customer on their automobile purchase. It may be difficult to read, since no bins have been created yet. See next section to learn how to do that. You can also do multiple variables at the same time, by simply adding to the TABLES statement, as follows:

PROC FREQ data=save.train;

TABLES dwnpmt trades ageavg vage;

Run;

To get basic descriptive statistics, try the following:

PROC Means data=save.train;

VAR dwnpmt;

Run;

**Creating Bins using Formats**

To get frequency tables, it makes sense to group continuous variables into bins so that the data can be meaningfully read. How large should the bins be? Should they be of equal size?

If you want to see the distribution to look for outliers or bad data or simply understand the distribution, it makes sense to use equal sized bins, to show the shape of the distribution. In this case, if the distribution is normal or close to it, you may find that a lot of observations are in some bins at the center of the distribution, while the tails are sparsely populated.

However, if we want to proceed to the next step, which is to do bivariate analysis, where we look at the impact of this variable on our dependent variable, we must have enough data in each bin to make it meaningful to conclude something about that range of values. So it is more important to have a reasonable volume in each bin (say 2% to 10% of the sample), and let the bin sizes vary as they may. So how do we determine the bin ranges?

1. Look at the cumulative frequencies in the frequency output to determine the groups (bins) that would contain about 2 to10% (say 5%) of the observations each.
2. Use those cutoffs to create a new format (a set of bins to help display the frequency tables) as follows:

Open the program file called **format.sas**. Create a new format (format name should be less than 8 characters, and must end in a character – not a number). Assume we are creating a format called DWNPMT, to associate with the variable of the same name. The program will look something like this.

**PROC FORMAT library=library;**

\*\*\*There will be predefined formats already in file.

\*\*\*Go to the end of the file and add the following:

VALUE **dwnpmt**

LOW – <0 = ‘Missing’

0 = “Zero”

0<-100 = “$1 to $100”

100<-500=”$100 to $500”

500<-800=”$500 to $800”

...

 ...

2400 <-HIGH

;

Run;

1. Run the format program above. This will overwrite the **FORMAT.SC2** file in the Formats folder, which is essentially a repository of formats that SAS can use when called upon. Note that the word **library** in the program above points to the **FORMATS** folder, as per the **start** program we ran at the beginning.
2. Create a frequency table for **dwnpmt** now with the format applied to it, as follows - In the program editor window, type:

PROC FREQ data=save.train;

TABLES dwnpmt;

FORMAT dwnpmt dwnpmt.;

/\*\*\* The first dwnpmt is the variable name. The second dwnpmt with the period after it refers to the format name to be applied \*\*\*\*/

Run;

Remember to always add formats to the format.sas file, rather than simply running a single new format definition in the editor window. If you run just one format definition, the format.sc2 file gets overwritten with just that one format, and the previous ones are not available anymore!

**Formatting multiple variables**

If two different variables (say dwnpmt and income) can use the same format (called dwnpmt), you might use the following syntax:

PROC FREQ data=save.train;

TABLES dwnpmt;

FORMAT dwnpmt income dwnpmt.;

Run;

If income uses a different format, also called income, the syntax is as follows:

PROC FREQ data=save.train;

TABLES dwnpmt;

FORMAT

dwnpmt dwnpmt.

income income.

;

Run;

Note that SAS does not care if the entire Format statement is on one line or split on 2-3 lines. The statement ends when the semicolon(“;”) is encountered.

**Associating Formats to Variables Permanently in a Dataset**

Using the Format statement as part of a PROC FREQ only associates a format to a variable for that particular run of the program. If you want to permanently associate a format with a variable, you can do the following:

PROC DATASETS library=save;

MODIFY **train**;

Format dwnpmt dwnpmt.;

Run;

If you now run a PROC CONTENTS on the **Train** dataset, you will seethat the variable *dwnpmt* is associated with the format **dwnpmt**. You can now run a PROC FREQ on that variable without the format statement, and it will still show up in the output formatted.

**Format Errors – Removing Formats**

What if a format is already associated with a variable in a dataset, and you do not wish to see it in a frequency table output? Try the following:

PROC FREQ data=save.train;

TABLES dwnpmt;

FORMAT \_ALL\_;

Run;

The Format \_ALL\_ statement tells SAS to ignore all formats and show the original data.

**The ERROR!:** If you have a dataset with formats associated to variables by someone else, and you do not have that format file and have not yet created any of your own, you will get error messages even with a PROC PRINT statement. SAS will look in the format.sc2 file for any formats associated with variables in the dataset before printing data or frequency tables. If it does not find the format.sc2 file at all, or sees the file but not the format name it is looking for, you will see an error message.

**Eliminating the Error**:

Option 1: Always make sure you have a **format.sc2** file in your FORMATS folder! It should contain all the formats that you need. If not, edit the **format.sas** program, add formats as needed, and run that program to update the format.sc2 file.

Option 2: The other option is to begin by clearing out all format associations from the dataset as follows:

PROC DATASETS library=save;

MODIFY **data1**;

Format \_ALL\_;

Run;

This will clear any permanent format associations, and you can use the file normally.

**Bivariate Analysis: Crosstabulations**

A crosstab is easy to create, and is a simply modification of the TABLES statement in PROC FREQ.

PROC FREQ data=save.train;

TABLES **dwnpmt\*Good**;

FORMAT dwnpmt **dwnpmt**. ;

Run;

Here, **good** is our binary dependent variable.

To eliminate row percentage values or overall percentage values, and output to html, you can do the following:

ODS html file=“D:\project\output\crosstab\_dwnpmt.html”;

PROC FREQ data=save.train;

TABLES **dwnpmt\*Good / nopercent norow**;

FORMAT dwnpmt **dwnpmt**. ;

Run;

ODS html close;