

SAS® Hash Objects: An Efficient Table Look-Up in the Decision Tree

Ying Liu, Toronto, ON, Canada

ABSTRACT

Table look-up is the one of the most important and frequently performed data processing operations. There are three commonly used techniques in the decision tree data processing; they are: IF-THEN-ELSE logic, MERGE with BY and FORMAT with PUT(). IF-THEN-ELSE, although conceptually very easy can require a lot of coding and CPU processing time; moreover, when the tree logic changes the maintenance of the tree can be an onerous task. MERGE with BY forces us to sort and re-sort the data; in addition, many passes through the data may be required. Sorting large data sets can increase the CPU and disk requirements significantly. Using SAS FORMATS with PUT() can reduce the CPU and program maintenance requirements, but they do not allow us to do compound key lookup.

The HASH Object, introduced in SAS 9.1, performs the standard operations of finding, adding and removing data dynamically in the memory. No sorting is required for the input data sets. It is created at run-time and will grow in memory as needed. In most applications the Hash Object will have better performance than the alternatives.

This paper will review the three common techniques (IF-THEN-ELSE logic, MERGE with BY and FORMAT with PUT().) and how they can be used to create a decision tree. Then the paper will focus on how to build and apply the hash look-up table in the decision tree data processing to achieve the following functions: grouping data, identifying missing data and transforming data.

Keywords: Hash, table look-up, decision tree

INTRODUCTION

A table lookup is a common data processing operation in processing a decision tree. The concept of a table lookup is when you have the known value of a variable in one table to look up the value of same variable in another table to return some related information, or simply to verify its existence in the table. A common example of a lookup table would be a state code (e.g. FL) and the state name (e.g. Florida). The simplest form of a table lookup is the use of the IF-THEN statement. IF-THEN statement is often the slowest table lookup methods although it is easy to code. The use of MERGE with BY, the use of FORMAT with PUT() and HASH Look-up Method substantially improves the data management and increases efficiency in both real time and CPU time reduction.

DECISION TREE

DECISION TREE CONCEPT

A decision tree (or tree diagram) is a decision support tool that uses a graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

In data mining, a decision tree is a predictive model; that is, a mapping from observations about an item to conclusions about its target value. More descriptive names for such tree models are classification tree (discrete outcome) or regression tree (continuous outcome). In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications.

In credit risk a decision tree is often used to help determine default rates.

DECISION TREE STRUCTURE

Ordinary tree structure

The ordinary tree consists of one root, branches and leaves. Please see the tree graph (Figure 1)

This is a binary tree. From the root upward, it has the two branches at each fork. At the end of each branch, there are one or two leaves.

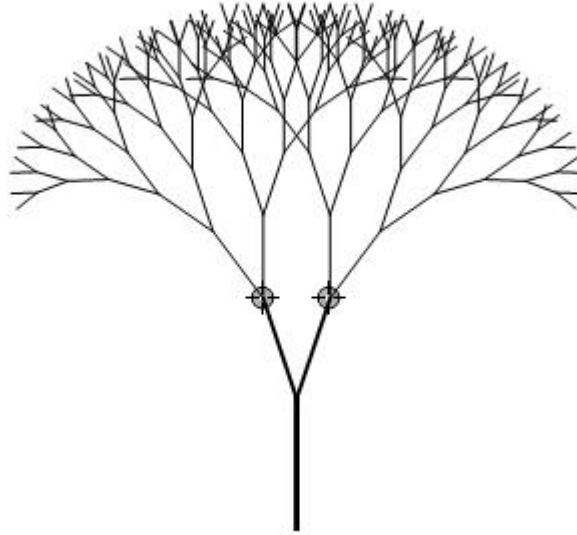


Figure 1: Ordinary Tree

Decision tree structure

The decision tree performs the same way as the ordinary tree. It consists of nodes denoted by circles and branches denoted by segments connecting the nodes.

Unlike the ordinary tree depicted as the root upward, the decision tree is usually drawn from left to right or beginning from root downward. The first node called a root is the starting point, while the end of the chain “root-branch-node-branch-...-node” is called a leaf. Each internal node grows one, two or more branches, corresponding to the values of the decision variables.

From Top to Bottom:

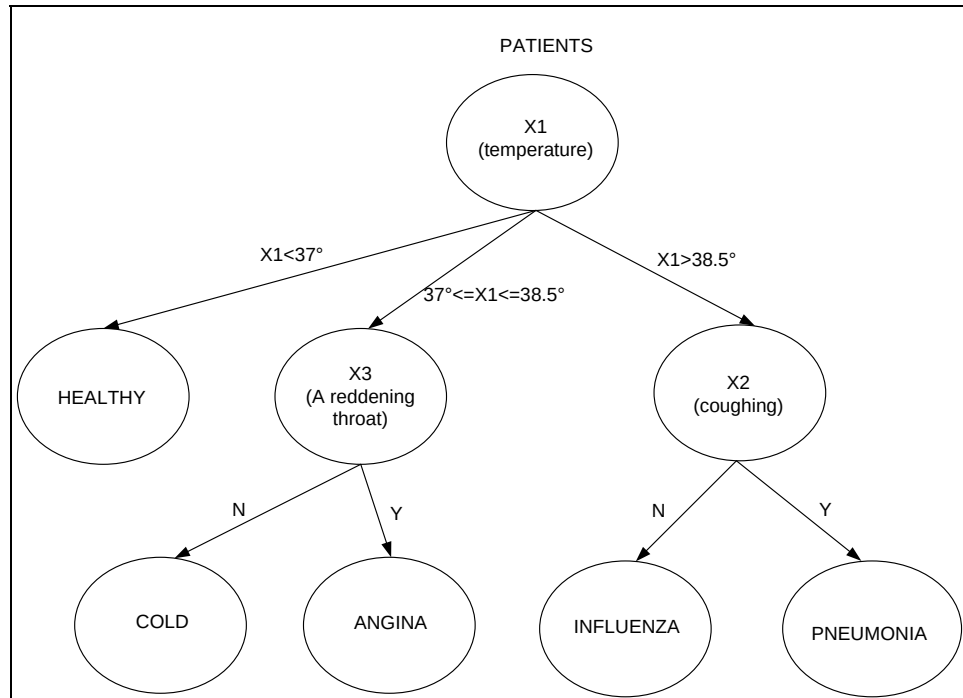


Figure2: Top-Down Tree

From Left to Right:

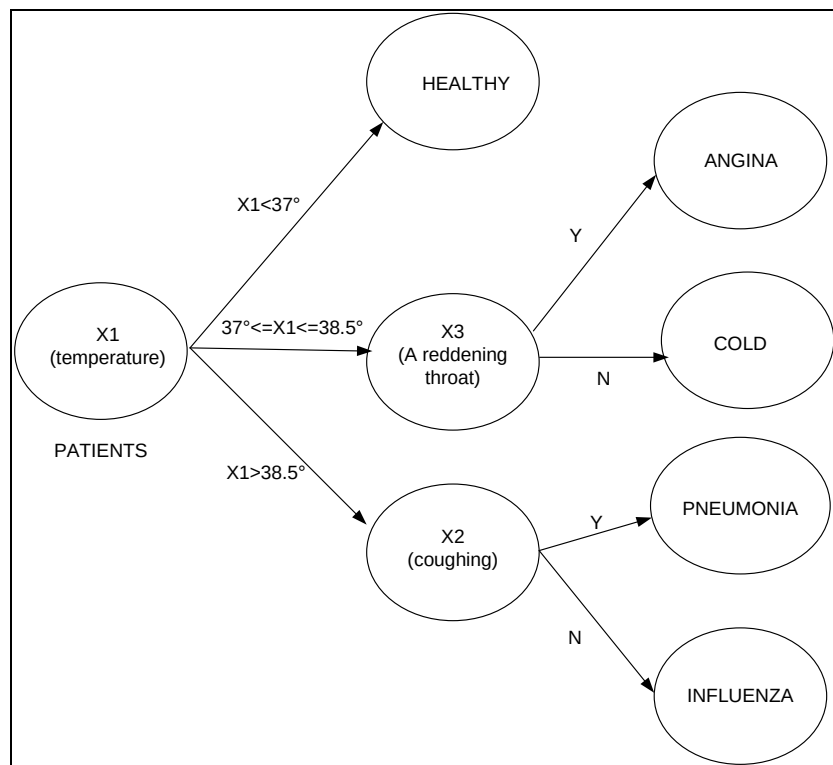


Figure 3: Left – Right Tree

INFORMATION TRANSFORMATION

In the above decision tree, there are 3 symptoms:

- Temperature
- Coughing
- A reddening throat

and 5 possible outcomes (leaf nodes):

- Healthy
- Angina
- Cold
- Pneumonia
- Influenza

The 3 symptoms are transformed into 3 independent decision variables:

- X1: temperature range; L – below 37; M – between 37 and 38.5; H – above 38.5.
- X2: coughing
- X3: a reddening throat

Based upon the values of the decision variable, the outcome is denoted by the dependent variable Y in the leaf node.

Branches are transformed into a set of logic statements. Each statement is obtained by passing the way from root to leaf. There are 5 logic assumptions:

- 1) if temperature is low, a person is healthy.
- 2) if temperature is medium, check if a patient has a reddening throat. If yes, the patient is diagnosed as having an angina;
- 3) if temperature is medium, check if a patient has a reddening throat. If no, the patient has diagnosed as having a cold.
- 4) if temperature is high, check if a patient is coughing. If yes, the patient has pneumonia.
- 5) if temperature is high, check if a patient is coughing. If no, the patient has influenza.

In the decision tree analysis, these assumptions are transformed into the following logic statements:

- 1) if x1 = 'L', then Y = 'HEALTHY'
- 2) if x1 = 'M' and x3 = 'Y', then Y = 'ANGINA'
- 3) if x1 = 'M' and x3 = 'N', then Y = 'COLD'
- 4) if x1 = 'H' and x2 = 'Y', then Y = 'PNEUMONIA'
- 5) if x1 = 'H' and x2 = 'N', then Y = 'INFLUENZA'

SAS TECHNIQUES IN DATA PROCESSING

There are 10 million records in the sample of clinic_data.

The partial of the clinic data can be viewed in the following diagram:

Patient_id	Temperature	reddening_throat	coughing
00000001	M	N	N
00000002	H	Y	N
00000003	H	N	N
00000004	L	N	N
00000005	H	N	N
00000006	L	N	N
00000007	L	N	N
00000008	M	N	N
00000009	H	Y	N
00000020	L	N	N
00000021	L	N	N
00000022	M	Y	N
00000023	H	Y	Y
00000024	H	Y	N
00000025	L	N	N

Table 1: Clinic_Data

METHOD1: IF – THEN – ELSE

IF statement is a logic processing. The recorded value is hard coded in one data step. It is easy for a programmer to code the IF statements.

The problem with multiple IF statements in the program is that SAS is forced to process each IF statement even if the condition is met in the first IF statement. However, with a very large data set, the use of multiple IF statements tremendously increase the CPU processing time.

SAS Code:

```
/* real time=10.24 secs, CPU=6.90 secs */
/* IF statements */
data clinic_ifThen;
  set pgm.clinic_data;
  length results $10;
  if temperature = 'L' then results = 'HEALTHY';
  if temperature = 'M' and reddening_throat = 'Y' then results = 'ANGINA';
  if temperature = 'M' and reddening_throat = 'N' then results = 'COLD';
  if temperature = 'H' and coughing = 'Y' then results = 'PNEUMONIA';
  if temperature = 'H' and coughing = 'N' then results = 'INFLUENZA';
run;
```

**Results using
If/Then logic**

The FREQ Procedure

Results	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
ANGINA	1580014	15.8	1580014	15.8
COLD	573633	5.74	2153647	21.54
HEALTHY	4285713	42.86	6439360	64.39
INFLUENZA	2491318	24.91	8930678	89.31
PNEUMONIA	1069322	10.69	10000000	100

Table 2: Results using IF statements

One of the improvements for the use of IF statement is to be conjunction with the use of ELSE statement. ELSE statement can substantially improve CPU time. The reason is when the first condition is met SAS does not process the ELSE condition. Moreover, the order of IF statement will impact efficiency as well. In this particular clinic data, since the majority patients are healthy, the IF statement temperature = 'L' should be placed at the beginning of the conditions.

The above code is rewritten using IF/THEN/ELSE statements. The results show the significant CPU reduction by 39% and real time process reduction by 32%.

SAS Code:

```
/* real time=6.95 secs, CPU=4.24secs */
* if/then/else statements **;

data clinic_ifThenElse;
  set pgm.clinic_data;
  length results $10;
  if temperature = 'L' then results = 'HEALTHY';
```

```

else if temperature = 'M'
then
do;
if reddening_throat = 'Y'      then results = 'ANGINA';
else if reddening_throat = 'N' then results = 'COLD';
end;
else
do;
if coughing = 'Y'              then results = 'PNEUMONIA';
else if coughing = 'N'         then results = 'INFLUENZA';
end;
run;

```

Results using If/Then/Else logic

The FREQ Procedure

results	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ANGINA	1580014	15.8	1580014	15.8
COLD	573633	5.74	2153647	21.54
HEALTHY	4285713	42.86	6439360	64.39
INFLUENZA	2491318	24.91	8930678	89.31
PNEUMONIA	1069322	10.69	10000000	100

Table 3: Results using IF/THEN/ELSE statements

However, the hard coding results are very difficult to maintain, especially in a complicated tree.

An alternative solution is to have a separate table to record the recorded values. It would be more useful and easily maintained.

This removes the logic in the data step and an issue of data maintenance, a task that is less error prone. In this case, a look up table with the 12 possible combinations from the above decision tree is created.

Temperature	coughing	reddening_throat	results
L	Y	Y	HEALTHY
L	Y	N	HEALTHY
L	N	Y	HEALTHY
L	N	N	HEALTHY
M	Y	Y	ANGINA
M	N	Y	ANGINA
M	Y	N	COLD
M	N	N	COLD
H	Y	Y	PNEUMONIA
H	Y	N	PNEUMONIA
H	N	Y	INFLUENZA
H	N	N	INFLUENZA

Table 4: Diagnosis

METHOD2: MERGE WITH BY

MERGE with a BY statement joins observations from two or more SAS data sets into a single observation in a new data set according to the values of a common variable listed in the BY statement. The data sets that are listed in the

MERGE statement must be sorted in order of the values of the common variables. This technique is demonstrated in the following DATA step.

There are two data sets used in the MERGE with BY process, diagnosis data set listed in the above diagram (Table 4: Diagnosis) and the source data clinic_data (Table1: clinic_data).

SAS code:

```
*** merge with by *;

proc sort data=diagnosis out=lookUp;
  by temperature coughing reddening_throat;
run;

proc sort data=pgm.clinic_data out=clinic_data;
  by temperature coughing reddening_throat;
run;

data clinic_merge_lookUp;
  merge   clinic_data  (in=inData)
         lookUp        (in=inLookup)
        ;
  by temperature coughing reddening_throat;
  if inData;
run;

/* keep the same key order in the original source data */
/* This sort step is optional */
proc sort data=clinic_merge_lookUp;
  by patient_id;
run;
```

Results using Merge with By

The FREQ Procedure

Results	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ANGINA	1580014	15.8	1580014	15.8
COLD	573633	5.74	2153647	21.54
HEALTHY	4285713	42.86	6439360	64.39
INFLUENZA	2491318	24.91	8930678	89.31
PNEUMONIA	1069322	10.69	10000000	100

Table 5: Results using MERGE with BY

The problem of the above approach is SAS must go through multiple sorts. Both data sets have to be sorted before the merge step. Sorting takes significantly CPU time especially when either or both data sets are large. It is also noted that both data sets have to be processed twice – once to sort and once to merge, unless your input data has already in the sort order required to merge.

METHOD3: FORMAT WITH PUT ()

Rather than the use of IF/THEN/ELSE in a single pass of the data, a SAS programmer can take advantage of the look up table (Table 4: diagnosis) that is created above and create a format to recode the variables.

A format is an instruction that SAS users to recode data values. SAS has internal formats supplies with SAS and the user defined formats. Both formats change the display of data values.

Using values in the table 4: diagnosis, a user defined format DESC is created using PROC FORMAT statement:

SAS Code:

```
proc format;
  value $Desc
    'LYY', 'LYN', 'LNY', 'LNN' = 'HEALTHY'
    'MYY', 'MNY' = 'ANGINA'
    'MYN', 'MNN' = 'COLD'
    'HYY', 'HYN' = 'PNEUMONIA'
    'HNY', 'HNN' = 'INFLUENZA'
  ;
run;
```

Now the new format DESC will be used to determine the outcomes in one data step:

SAS Code:

```
* format with put() *;
data clinic_format;
  set pgm.clinic_data;
  length lookUp_key    $3;
  length results        $10;
  lookUp_key = temperature || coughing || reddening_throat;
  results = put(lookUp_key, $desc.);
run;
```

Results using FORMAT with PUT()

The FREQ Procedure

Results	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ANGINA	1580014	15.8	1580014	15.8
COLD	573633	5.74	2153647	21.54
HEALTHY	4285713	42.86	6439360	64.39
INFLUENZA	2491318	24.91	8930678	89.31
PNEUMONIA	1069322	10.69	10000000	100

Table 6: Results using FORMAT with PUT()

Recording data using FORMAT with PUT() has a number of advantages:

1) Compared to IF/THEN statement:

- Less coding: once a format is created, the recoded value is created through the PUT() function. The data step code is much shorter, easier to follow. This efficiency becomes significantly as the number of items in the lookup list increases. There is only one executable PUT() statement rather than one for each comparison.
- Easy to maintain: when a recoded value has to be changed, we only need to make a change in one location which is the list of values in the proc format statement without touching data steps.

2) Compared to Merge with By:

A lookup is created through PROC FORMAT statement. Only one data step is processed once without sorting. Both CPU and IO time are dramatically reduced.

Creating a CNTLIN data set can enhance through this method. Please see "Liu, SAS Formats, More Than Just Another Pretty Face" for the details.

METHOD4: HASH LOOK-UP

A hash table is created and sized in memories during the execution of the data step. When the data step ends, the hash object disappears.

A hash search method is an index search. It uses a fast, non-linear search to index unsorted data with a key.

There are multiple ways to create and use a hash object. For a thorough discussion and the used of the hash object, please see any of the Dorfman papers listed in the reference.

SAS code:

```
*** hash table **;  
data clinic_hash (drop=rc);  
  declare hash lookUp();  
  rc=lookUp.defineKey('temperature','coughing','reddening_throat');  
  rc=lookUp.defineData('results');  
  rc=lookUp.defineDone();  
  do until(eof1);  
    set diagnosis end=eof1;  
    rc=lookUp.add();  
  end;  
  do until(eof2);  
    set clinic end=eof2;  
    call missing(results);  
    rc=lookUp.find();  
    output;  
  end;  
  stop;  
run;
```

Results using Hash Look-Up

The FREQ Procedure

results	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ANGINA	1580014	15.8	1580014	15.8
COLD	573633	5.74	2153647	21.54

HEALTHY	4285713	42.86	6439360	64.39
INFLUENZA	2491318	24.91	8930678	89.31
PNEUMONIA	1069322	10.69	10000000	100

Table 7: Results using HASH Look-UP

TIME EFFICIENCY COMPARISON

Efficiency is measured by the time saving in the codes processing.

The test randomly selects 10 million records and ran IF statement, IF/ELSE statement, MERGE with BY method, the use of FORMAT with PUT and the Index search of HASH object. The results of the real time cost and CPU time cost can be seen in the below table 8: Processing Time Compare Results. It is not surprising that in this simple decision tree with only 12 possible outcomes, the use of IF/ELSE is relatively efficient. However, a SAS programmer should be aware of that the use of IF/ELSE is the logic processing in one data step, the recorded values are hard coded in the data step. With the number of outcomes increasing, IF/ELSE will increase the error caused by the hard coding results and codes become very difficult to maintain.

The use of MERGE with BY, FORMAT with PUT and HASH object store the recorded values in a look up table. They have a less data maintenance problem. Their coding is shorter and more clarity than the use of IF/ELSE statement. Since their search methods are so different, from Merge with BY method to FORMAT with PUT method: real time is reduced by 43% and CPU is reduced by 50%. From FORMAT with PUT method to HASH look up method: real time is reduced by 55% and CPU is reduced by 56%. Therefore HASH look-up is the most efficient table look up method.

Records:
10,000,000

	IF Statement	IF/ELSE	Merge with By	Format with PUT	Hash
real time	10.24 seconds	6.95 seconds	24.99 seconds	14.36 seconds	6.50 seconds
CPU time	6.90 seconds	4.24 seconds	28.69 seconds	14.29 seconds	6.22 seconds

Table 8: Processing Time Compare Results.

HASH TABLE IN THE COMPLICATED DECISION TREES

The above example uses a very simple decision tree. Unfortunately, in the real world of credit risk, the decision trees are not so simple. The following discussion will focus on a typical type of a multiple level decision tree that an analyst in the credit risk world often encounters. In this example, the analyzed population is segmented into a four – level decision tree. Please see the below figure 4: Multiple Levels Decision Tree.

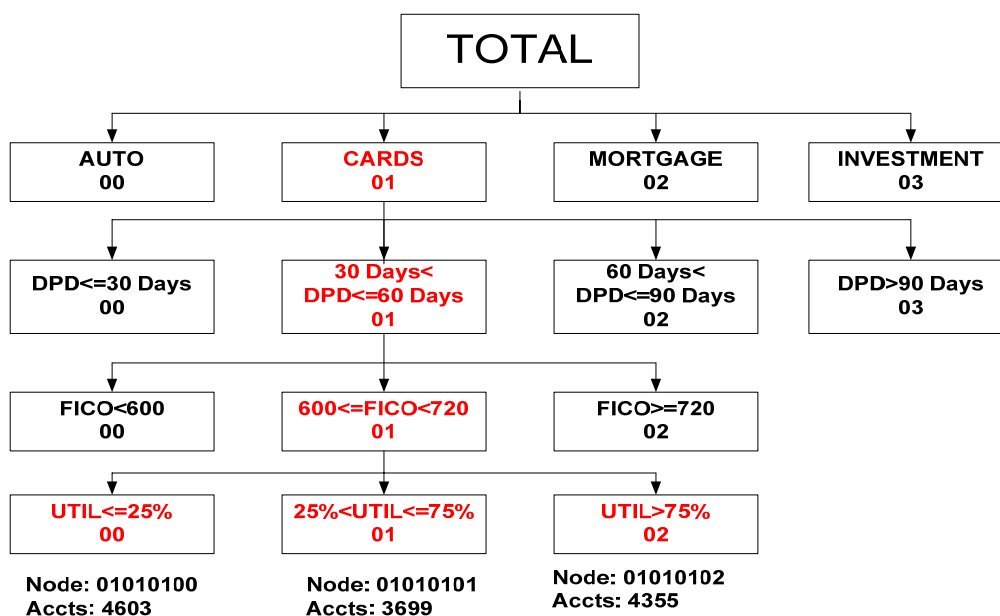


Figure 4: Multiple Levels Decision Tree

INFORMATION GATHERING

In the above decision tree, it is not hard to find out there are 4 dependent variables there. They are: portfolio, days past due (DPD), fico score (FICO) and utilization (UTIL).

And the decision tree information is processed from top downwards and split into 4 levels.

- Level 1: total population is split by 4 types of portfolio.
- Level 2: each portfolio is split by the 4 different ranges of DPD.
- Level 3: each DPD is split by 3 different range of FICO.
- Level 4: each FICO is split by 3 different ranges of UTIL.

Outcomes: This complex tree results in 144 combinations (4 portfolio * 4 DPD * 3 FICO * 3 UTIL)

INFORMATION PROCESSING

As depicted in the figure 4, the dependent variables days past due, fico score and utilization are continuous variables and portfolio type is a string variable. Please see partial raw data in the below table 9.

product	days_past_due	bureau_score	Utilization
AUTO	90	0	0.10
AUTO	.	850	0.20
AUTO	20	800	0.40
AUTO	85	600	0.45
AUTO	31	650	0.75
AUTO	70	550	0.80
AUTO	120	500	0.80

Table 9: Raw Data

First all dependent variables need to be recorded into the 2-digits variables – recoding process; then in the final visual representation the recorded values need to be decoded with the representation of the meaningful text words – decoding process. This recoding value process and decoding process can be accomplished through PROC FORMAT. Please see the below table 10 – table 13.

Recoding Values	Decoding Values
<pre>proc format; value \$ prodFmt 'AUTO' = '00' 'CARDS' = '01' 'MORTGAGE' = '02' 'INVESTMENT' = '03' other = '99' ;</pre>	<pre>proc format; value \$ prodDesc 00 = 'auto' 01 = 'cards' 02 = 'mortgage' 03 = 'investment' 99 = 'others' ;</pre>

Table 10: portfolio recording and decoding values

Recoding Values	Decoding Values
<pre>proc format; value delfmt . = '00' 0 - 29 = '00' 30 - 59 = '01' 60 - 89 = '02' 90 - high = '03' ;</pre>	<pre>proc format; value \$delDesc 00 = 'Not Due' 01 = '30 Days Past Due' 02 = '60 Days Past Due' 03 = '90 Days+ Past Due' 99 = 'Error' ;</pre>

Table 11: DPD recording and decoding values

Recoding Values	Decoding Values
<pre>proc format; value ficoFmt . = '00' 0 - <600 = '00' 600 - <720 = '01' 720 - high = '02' ;</pre>	<pre>proc format; value \$ ficoDesc 00 = 'Missing or 0<=Fico<600' 01 = '600<=Fico<720' 02 = 'Fico>=720' 99 = 'Error' ;</pre>

Table 12: FICO recording and decoding values

Recoding Values	Decoding Values
<pre>proc format; value utilFmt low - 0.25 = '00' 0.25< - 0.75 = '01' 0.75 - high = '02' ;</pre>	<pre>proc format; value \$ utilDesc 00 = 'Low Utilization' 01 = 'Medium Utilization' 02 = 'High Utilization' 99 = 'Error' ;</pre>

Table 13: UTIL recording and decoding values

LOOK UP TABLE

Clearly, in this complex decision process IF/THEN/ELSE is unattainable. Creating a look up table with the 144 outcomes becomes more feasible. In this particular decision tree analysis, all portfolios happen to have the same range of DPD, FICO and UTIL. It is easy to create the look up table with multiple Do Loops. The following codes demonstrate how to create this look up table with 144 outcomes.

SAS code:

```
data lookup (keep=node node_desc);
  length i 3.;
  i = 0;
  length comma $3.;
  comma = " , ";
  length node $8;
  length node_desc $80.;
  do prodIndex = '00', '01', '02', '03';
    do dpdIndex = '00', '01', '02', '03';
      do ficoIndex = '00', '01', '02';
        do utilIndex = '00', '01', '02';
          node = prodIndex || dpdIndex || ficoIndex || utilIndex;
          node_desc = put(prodIndex, $prodDesc.) || comma ||
                     put(dpdIndex, $delDesc.) || comma ||
                     put(ficoIndex, $ficoDesc.) || comma ||
                     put(utilIndex, $utilDesc.);
          i = i + 1;
          output;
        end;
      end;
    end;
  end;
run;
```

The partial results of the look up table:

node	node_desc
00000000	auto , Not Due , Missing or 0<=Fico<600 , Low Utilization

00000001	auto	, Not Due	, Missing or 0<=Fico<600	, Medium Utilization
00000002	auto	, Not Due	, Missing or 0<=Fico<600	, High Utilization
00000100	auto	, Not Due	, 600<=Fico<720	, Low Utilization
00000101	auto	, Not Due	, 600<=Fico<720	, Medium Utilization
00000102	auto	, Not Due	, 600<=Fico<720	, High Utilization
00000200	auto	, Not Due	, Fico>=720	, Low Utilization
00000201	auto	, Not Due	, Fico>=720	, Medium Utilization
00000202	auto	, Not Due	, Fico>=720	, High Utilization
00010000	auto	, 30 Days Past Due	, Missing or 0<=Fico<600	, Low Utilization

Table 14: Look –Up Table

HASH LOOK UP PROCESS:

SAS code:

```
data finalData(drop=rc );
  length node_desc $80;
  declare hash h(dataset: 'work.lookup');
  h.defineKey('node');
  h.defineData('node_desc');
  h.definedone();
  call missing(node_desc);
  do until (eof1);
    set rawdata end=eof1;
    length node $8;
    node = put(trim(product), $prodFmt.) ||
           put(days_past_due, delfmt.) ||
           put(bureau_score, ficofmt.) ||
           put(utilization, utilfmt.);

    rc = h.find();
    output;
  end;
  stop;
run;
```

Results when running finalData set using PROC FREQ.

node_desc	product	days_past_due	bureau_score	Utilization	node
auto , 90 Days+ Past Due , Missing or 0<=Fico<600 , Low Utilization	AUTO	90	0	0.1	00030000
auto , Not Due , Fico>=720 , Low Utilization	AUTO		850	0.2	00000200
auto , Not Due , Fico>=720 , Medium Utilization	AUTO	20	800	0.4	00000201
auto , 60 Days Past Due , 600<=Fico<720 , Medium Utilization	AUTO	85	600	0.45	00020101
auto , 30 Days Past Due , 600<=Fico<720 , Medium Utilization	AUTO	31	650	0.75	00010101
auto , 60 Days Past Due , Missing or 0<=Fico<600 , High Utilization	AUTO	70	550	0.8	00020002
auto , 90 Days+ Past Due , Missing or 0<=Fico<600 , High Utilization	AUTO	120	500	0.8	00030002
auto , 90 Days+ Past Due , Missing or 0<=Fico<600 , High Utilization	AUTO	100		0.9	00030002
auto , 30 Days Past Due , 600<=Fico<720 , High Utilization	AUTO	58	620	0.95	00010102
auto , 90 Days+ Past Due , Missing or 0<=Fico<600 , High Utilization	AUTO	300	500	0.99	00030002

cards	, Not Due	, Fico>=720	, Low Utilization	CARDS		720	0	01000200
-------	-----------	-------------	-------------------	-------	--	-----	---	----------

Table 15: Final Data Representation

CONCLUSIONS

There are different techniques that can be applied when a user needs to recode values. In the simple decision tree such as one or two level decision tree, when the outcomes are very limited, IF/THEN/ELSE might be a fast and efficiency method to create recorded values. Since the use of if/then/else is a hard coding process, it creates a huge data maintenance problem. As the list of recorded value increases, it becomes more useful and maintainable to have a look up table to hold these recorded values.

With the table look up, MERGE with BY requires the input data sets to be sorted or indexed. FORMATS with PUT() can be used as a look up table method on large non-indexed data sets, but takes up the large amounts disk space. A HASH table is a memory – index search, faster than using a FORMAT, takes less memory and less amounts disk space.

REFERENCES:

Dorfman, Paul and Wyverman, Koen "Data Step Hash Objects as Programming Tools". *Proceedings of the Thirtieth Annual SAS Users Group International Conference*, 2005.

Dorfman, Paul and Wyverman, Koen "Hash Component Objects: Dynamic Data Storage and Table Look-Up". *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*, 2004.

Dorfman, Paul "Table Lookup By Direct Addressing: Key-Indexing, Bitmapping, Hashing". *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference*, 2001.

Liu, Ying "SAS Formats: More Than Just Another Pretty Face". *Proceedings of Fifteenth Annual Southeast SAS Users Group Conference*, 2007

ACKNOWLEDGEMENTS

Peter Eberhardt provided me with the valuable insights during the creation of the paper. It has been a great pleasure to learn from him.

CONTACT INFORMATION

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

Ying Liu
HSBC Financial
3381 Steeles Avenue East, Suite 300
Toronto, Ontario M2H 3S7 Canada
Work Phone: (416) 443-3699
Fax: (416) 443-3749
E-mail: ying.x.liu@hsbc.ca

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.