SESUG Proceedings (c) SESUG, Inc (http://www.sesug.org) The papers contained in the SESUG proceedings are the property of their authors, unless otherwise stated. Do not reprint without permission.
SEGUG papers are distributed freely as a courtesy of the Institute for Advanced Analytics (http://analytics.ncsu.edu).

Paper CS-058

Rule based filtering - Categorizing unwanted inputs

Richard A. DeVenezia, Independent Consultant, Remsen, NY

Abstract

Web based survey questions typically request a respondent make one or more selections from a prepared list of valid values. However, it may be the case that a portion of the survey is designed to collect free form input via text boxes. There is no guarantee that the input will be valid for the context of the question, or that the respondent will be particularly helpful in their data entry.

This paper will only address the issue of eliminating nonsense inputs (noise), by way of pattern detection and the use of if/then rule sets whose parameters are stored in an Excel workbook.

The inputs that pass the noise check are passed to an ETL process that performs further processing that classifies inputs into additional categories of *matched (to a master list)* and *unmatched*. This processing is beyond the scope of this paper.

Keywords: Survey, Filtering, Noise, Rules, Control data, Code generation

Survey data

Consider an English language survey that gathers information from book purchasers. One section of the survey allows the participant to type in the titles of up to ten books they might purchase in the next three months, the names of ten titles their friends mentioned to them as a good read and the title of the last book they read. There is no additional prompting so that the data gathering is unbiased.

A good survey campaign must select participants willing to invest time in taking the survey. In many situations the participants are offered a reward of some sort. Perhaps a monetary award, a coupon, or points to be accumulated for application to a more complex award system.

There are some participants who will enter nonsense values (noise) in order to speed their way through the survey to the reward. The goal of this paper is to show a technique for detecting these values in a straight forward and managed way.

For purposes of discussion the input data will be named Item1 through Item21. Presume N prior similar surveys have occurred, so that there is a body of prior item values that can be examined.

Core Logic

The core logic of the noise detection, or noise detector, is admittance after vetting. When an item value is not considered noise by <u>any</u> of the value tests, the item is allowed to pass through to the remainder of the ETL process. Logically this can be stated as:

```
if
  item-value fails(is-noise-assertion-1)
... and ...
  item-value fails(is-noise-assertion-N)
then
  item-value is not noise
```

The second logic flow form is to state the inverse:

```
if item-value passes(is-noise-assertion-1) then goto IsNoise;
...
if item-value passes(is-noise-assertion-N) then goto IsNoise;

* did not pass any is-noise assertions, thus is not noise;
IsNotNoise:
  Continue; * with next item;
IsNoise:
  * do something with noise value;
  Continue; * with next item;
```

Noise detection (and subsequent matching algorithms) will operate best when the item-values are transformed to a level playing field. In this paper the term *flattened* will mean the items have been stripped of excessive repeated leading punctuation, lowercased, stripped of non-alpha, non-digit, non-space characters and had adjacent spaces reduced to one. Additionally, some 8-bit ANSI characters will have been translated to 7-bit ASCII.

Core implementation

The core of the implementation uses the second logic flow form. A SAS® DATA Step performs the logic checks. Noise values are output to a separate table for reporting purposes. An ARRAY and DO loop are used for examining each item of the raw input:

```
* generate DATA Step statements that will be used in following DATA Step;
filename RuleStmt catalog 'work.codegen.noise_testing_stmts.source';
%Rules_Processor (Statements=RuleStmt);

DATA
  Items
  Items_Noise (keep=id _name_ _value_ flat reason)
;
  SET Items_raw;
  ...
  ARRAY Items Item1-Item21;
  do index = 1 to DIM(Items);
    item = Items[index];
    if item = '' then continue;
    ... remove excessive leading punctuation ...

    %Noise_Detector (GeneratedRuleStatements=RuleStmt);

    continue;

    NoiseDetected:
    do;
      %* change item-value to noise reason;
      %* reason has been set within Noise_Detector;
      _name_ = VNAME (items[index]);
      _value_ = items[index];
      flat = item;
      items[index] = cats ('*',reason,'*');

      output Items_Noise;
    end;
  end;
  output Items;

run;
```

The tricky part here is that the %Noise_Detector uses previously generated DATA Step code that contains

```
if ... then GOTO NoiseDetected;
```

statements. Note that when an item has been deemed as noise its value is changed to **reason**. This is done because the subsequent ETL code that deals with item-values will ignore any that begin with an asterisk.

The innards of the Rules_Processor, Noise_Detector and the code they generate are covered later.

Remove excessive leading punctuation

Consider "!!! Huckleberry Finn". On one hand, there may exist some title with several leading exclamation points, but probably none with more than eight. The following code will strip all but eight repeats of some repeated leading punctuation:

```
 if ANYPUNCT(item) = 1 then
        if VERIFY(item,SUBSTR(item,1,1)) > 8 then do;
          PUT "NOTE" ":&SYSMACRONAME.: "
            "Removing excessive leading punctuation " item=;
          item = SUBSTR(item,VERIFY(item,SUBSTR(item,1,1)) - 8);
        end;
```

%Noise_Detector

The noise detector macro generates a DATA Step code that prepares the item values, and then performs a series of procedural tests, followed by an INCLUDE of a series of generated if/then statements created from rules data. The rules data is stored in an external Excel spreadsheet.

Data preparation

Flattening

A series of nested function calls performs the bulk of the flattening. The COMPRESS invocation utilizes the modifiers argument so as to keep only alphabetics, digits and spaces. This is followed by the first procedural check. Notice how the reason is set before performing and acting on the test:

```
item = TRANSLATE(item,'ec','éç');
item = COMPBL(LEFT(COMPRESS(LOWCASE(item),,'adsk')));

reason = "flattened item is blank";
if item = '' then &OnIsNoise_Statement.;
```

`OnIsNoise_Statement` is a macro variable local to and used throughout the Noise_Detector. The consistent use of this macro variable is a proviso for managing future changes in noise handling. At present the statement is a simple goto:

```
%local OnIsNoise_Statement;
%let OnIsNoise_Statement = goto NoiseDetected;
```

A future change might look like this:

```
%let OnIsNoise_Statement = %STR
  ( do;
     ...some new statements...
     goto NoiseDetected;
    end;
  );
```

Noise categories

Each item determined to be noise is categorized according to the test that the item failed to pass. Consider the case of a flattened item that has become a blank value:

```
reason = 'flattened item is blank';
if item = '' then &OnIsNoise_Statement.;
```

%Noise_Detector (part 2)

The following are the additional procedural checks in the detector. Only one character:

```
reason = "1 char";
if length(item) = 1 then &OnIsNoise_Statement.;
```

Four or more repeats of a character, or 1st or 2nd character repeated exclusively:

```
reason = "repeated char";

length _c1 _c2 $1; drop _c1 _c2;
_c1 = SUBSTR(item,1,1);
_c2 = SUBSTR(item,2,1);

/* all same as 1st character */
if (length(item) > 3 and compress(item,_c1) = ' ')
or
/* all same as 1st character, except one */
(length(item) > 5 and length(compress(item,_c1)) = 1)
or
/* all same as 2nd character, except one */
(length(item) > 5 and length(compress(item,_c2)) = 1)
then
  &OnIsNoise_Statement.;
```

All digits:

```
reason = "all digits (five or more)";
if length(item) > 4 and compress(item,,'d')='' then &OnIsNoise_Statement.;
```

First two characters alternated repeatedly:

```
reason = "alternation";
if length(item) > 4
and TRANWRD(item,SUBSTR(item,1,2),' ')=' ' then &OnIsNoise_Statement.;
```

Very long values:

```
reason = ">100 char";
if length(original) > 100 then do;
  original = compbl(original);
  if length(original) > 100 then &OnIsNoise_Statement.;
end;
```

A number of the tests were layered into the detector as empirical study of ongoing surveys occurred.
The last part of the noise detector makes use of two design elements; if/then statements generated from control data:

```
%include &GeneratedRuleStatements;
```

and a custom format [2] for blacklisting:

```
reason = "blacklist";
```

```
if put (item,$ItemBlacklistAction.) = 'delete' then &OnIsNoise_Statement.;
```

`GeneratedRuleStatements` is macro variable (and parameter of %Noise_Detector) whose value is a fileref that points to source code written by another macro.

%Rules_Processor

The Rules_Processor reads the control data and writes statements for use by Noise_Detector. There are data for two types of rules.

Rule type 1 data – Blacklist

A list of values used for exact matching. If an item matches any of the values in the black list, the item will be disregarded. The blacklist is a good place to filter out values that are distinctly unique enough to not be easily matched with Rules.

	A
1	banned_input
2	asldkfj
3	bdfzv
4	cjcnj
5	cjg
6	cjnj

Read in the control data:

```
proc import
  datafile="Noise Detection Criteria.xls"
  dbms=excel
  replace
  out=BlackList(keep=banned_input where=(banned_input ne ''))
;
  sheet=BlackList;
  getnames=yes;
...
```

and create the custom format $ItemBlacklistAction. with it:

```
data BlackList;
  set BlackList;
  retain fmtname '$ItemBlacklistAction';
  rename banned_input = start;
  label = 'delete';
run;

proc format cntlin=BlackList;
run;
```

The custom format is used in Noise_Detector as show above at [2]

Rule type 2 data – Assertions

The notion of 'is-noise-assertion-1" was introduced on page 1. This screen-shot shows a part of the first two rules control data that the code generator will utilize.

	A	B	C	D	E	F	
1	RuleNumber	Reason	Role	Comparator	Criteria1	Criteria2	Criteri
2							
3	1	none	assert	in:	none	nothing in	nothing
4	1		assert	in	na	nonr	nothing
5	1		assert	in	nothing comes to min	the page cannot be found	
6							
7	2	dont know	assert	index	d0nt	i am not sure	not sur
8	2		assert	index	cant think	dont remem	
9	2		assert	in:	dont	i dont	idont

Rules

The values entered for rules are for matching and for vetoing a match
The general logic encapsulated in the rules data is thus:

IF
((assertion-1 [or assertion-2] ... [or assertion-N]) is TRUE)
and (NOT (veto-1 [or veto-2] ... [or veto-N]) is TRUE)
THEN
match has occurred.

Assertion criteria are combined with the comparators to create DATA Step statements that look like the following:

```
if
  (
     INDEX(item,criteria-1.a)
  or item IN: (criteria-2.a, criteria-2.b, criteria-2.c)
  )
and
  NOT (
     item IN: (criteria-3.a, criteria-3.b)
  or INDEX(item, criteria-4.a)
  or COMPARE(item,criteria-5.a) or COMPARE(item,criteria-5.b))
  )
then
  goto NoiseDetected;
```

RuleNumber

Rule number is used to group the rules assertions and vetoes

Reason

A quick explanatory description of the rule.
Place a comment mark, #, at the front of the reason to temporarily turn off a rule.

Role

Assert - The role of the criteria in this row is to assert that a match has been made. Each RuleNumber generally has one assertion, if more than one is entered, it is possible that the combined assertion criteria will be overly broad in matching. Each assertion is combined with OR logic.
Veto - The role of the criteria in this row is to veto a match that has been made. Vetoing is most often used when a broad assertion matches input that should not have been matched. Each veto is combined with OR logic.

Comparator

The type of pattern matching to perform with the criteria. There are three comparators handled by the code generator.
INDEX - The criteria can be found anywhere in the item. When multiple criteria are listed across the row, they are combined using OR logic.
IN: - The criteria must be found at the beginning of the item. When multiple criteria are listed across a row, they are implicitly combined using OR logic.
COMPARE - The criteria must be an exact match of the item. When multiple criteria are listed across a row, they are combined using OR logic.

Criteria1 - Criteria20

List each criteria of the rule function type across the row in separate cells.

Rules, rules and more rules

The use of an Excel spreadsheet for the repository of rules parameters allows it to be managed by domain experts who may not be SAS experts. Rules can be easily added, removed, extended or temporarily disabled. New rules can be added as a domain expert observes new noise patterns that occur in newer surveys.

One type of pattern that occurs (and is hard to blacklist) is *keysurfing*. This is when the noise input is made by typing adjacent keys, perhaps by running them left to right. A keysurfing noise value might contain “qwerty”, “asdf”, “zxcv”, “poiu”, “lkjh”, or “mnbv”. Sometimes a value that matches a keysurfing sequence (as noise) is actually not noise. In this situation the veto role is used to enter parameters that match the non-noise value. The vetoing match would override the initial noise detection.

%Rules_Processor (2)

Read in the control data:

```
  proc import
    datafile="Noise Detection Criteria.xls"
    dbms=excel
    replace
    out=NoiseCriteria
        ( keep=RuleNumber Reason Role Comparator Criteria:
          where=(RuleNumber>0)
        )
  ;
    sheet=Rules;
    getnames=yes;
...
```

Write logic statements for inclusion in a DATA Step. Only key excerpts are show here:

Introduce the rule by code generating the opening IF:

```
...
    if first.RuleNumber then do;
      qtext = cats ("'",reason,"'");
      put
      /// '%* ' RuleNumber= ';'
      // "reason = " qtext ';'
      // "if "
      ;
    end;
```

Introduce the roles opening parenthesis:

```
    if first.Role then do;
      if Role = 'veto' then
        put @3 'AND NOT';

      put @3 '( /* ' role '*/';
    end;
```

Logically chain the roles with OR:

```
    if not first.Role then
      put @5 'OR' @;
```

Generate the bulk of the IF statement – the portion that utilizes the criteria
Example: item IN ('one', 'two', 'three'):

```
if Comparator = 'in' then do;
  put @8 'item in' / @9 '(' @;
  do i = 1 to dim(Criteria) while (Criteria[i] ne '');
    if i > 1 then put;
    qtext = cats("'",Criteria[i],"'");
    put @11 qtext @;
  end;
  put ')';
end;
```

Example: item IN: ('one', 'two' 'three'):

```
else
if Comparator = 'in:' then do;
  put @8 'item in:' / @9 '(' @;
  do i = 1 to dim(Criteria) while (Criteria[i] ne '');
    if i > 1 then put;
    qtext = cats("'",Criteria[i],"'");
    put @11 qtext @;
  end;
  put ')';
end;
```

Example: INDEX(item,'one') or INDEX(item,'two') or INDEX(item,'three'):

```
else
if Comparator = 'index' then do;
  do i = 1 to dim(Criteria) while (Criteria[i] ne '');
    if i>1 then put @5 'OR' @;
    put @8 'index ( item, ' @;
    qtext = cats("'",Criteria[i],"'");
    put qtext @;
    put ')' ;
  end;
end;
```

Example: COMPARE(item,'one') or COMPARE(item,'two') or COMPARE(item,'three'):

```
else
if Comparator = 'compare' then do;
  do i = 1 to dim(Criteria) while (Criteria[i] ne '');
    if i>1 then put @5 'OR' @;
    put @8 '0 = compare ( item, ' @;
    qtext = cats("'",Criteria[i],"'");
    put qtext @;
    put ')' ;
  end;
end;
```

Finish with the roles closing parenthesis:

```
if last.Role then
  put @3 ')';
```

Place the noise handler dispatch at the end of a rules code-gen:

```
if last.RuleNumber then
  put 'then' / @3 '&OnMatch_DoStatement. ;' ;
```

Actual production code would have additional preliminary checks to ensure certain consistencies and constraints on the rules data.

Conclusion

Noise detection is an important part of ETL processes. Managing a large number of noise detection rules can be accomplished using control data. DATA Step code generation can be a complex task, but management benefits outweigh the complexity.

Contact Information

Richard A. DeVenezia
9949 East Steuben Road
Remsen, NY 13438
(315) 831-8802
http://www.devenezia.com/contact.php

Richard is an independent consultant who has worked extensively with SAS products for over fifteen years. He has presented at previous SUGI, NESUG and SESUG conferences. Richard is interested in learning and applying new technologies. He is a SAS-L Hall of Famer and remains an active contributor to SAS-L.

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.

This document was produced using OpenOffice.org Writer.