SESUG Paper AT-218-2019

Using PROC FCMP to Implement Anomaly Detection Method

Tsung-hsun Tsai, Research League, LLC; Yung-chen Hsu, GED Testing Service

ABSTRACT

The essential purpose of PROC FCMP, the SAS® Function Compiler procedure, is to allow the creation of user defined functions or CALL routines for DATA steps or SAS procedures. Those functions can also be shared with other SAS programmers to facilitate code reuse and increase productivity. For advanced users, it can serve as a wrapper function for PROC PROTO to enable incorporating C language structures, types, and functions into SAS®. In addition, for users who do not have access to SAS/IML®, PROC FCMP also provides CALL routines for performing typical matrix operations, such as multiplication, transpose, and inverse. In this paper, we use a simple anomaly detection method for screening field test items as an example to illustrate the use of PROC FCMP and its capability of performing matrix operations. In testing service organizations, evaluating field test items in the posterior human review is an elaborate inspection process during the test development. This work demonstrates a method to automatically flag potential flawed items by using PROC FCMP to conduct matrix and low-level array operations.

INTRODUCTION

In testing service organizations, newly-developed test items will need to be tried out on actual examinees prior to operational use. Field test statistics are collected and calculated for the following posterior human review to adjudicate the operational status. The review process is time-consuming and labor-intensive. To alleviate the workload, we implemented an anomaly detection method based on a simple multivariate Gaussian distribution modeling approach to narrow down the review list of items automatically. PROC FCMP is used to conduct matrix and low-level array operations of this approach. SAS does have a SAS/IML® language to solve matrix computation, which is relatively succinct and contains many functions. However, the license of SAS/IML® requires an extra cost while the FCMP procedure already comes with SAS/BASE®. The formula can be decomposed to use matrix algebra. Each matrix operation can be performed in a PROC FCMP CALL routine.

ANOMALY DETECTION

Anomaly detection is the identification of items which differ from the majorities. There are many methods in literature with different performance and cost in a variety of domains. Here we consider a simple multivariate Gaussian distribution approach. If we consider a certain expected distribution of the normal items, then abnormal or unusual items are those that deviate by a certain threshold from the distribution. Certain attributes are collected or derived for building the anomaly detection model. Given a training set x_i for each feature i, we will first fit a model to the data's distribution. We estimate the Gaussian distribution for each of the features x_i of m items and find parameters μ_i and σ_i^2, where

$$\mu_i = \frac{1}{m}\sum_{m} x_i$$

and

$$\sigma_i^2 = \frac{1}{m}\sum_{m}(x_i - \mu_i)^2$$

Now, for a given feature vector x of a particular item, we then compute the probability with the given multivariate Gaussian distribution

$$p(x) = \prod_{i} \frac{1}{\sqrt{2\pi}\sigma_i} exp\left(\frac{-(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

If p is small, it means this particular x has a low probability and is more likely to be an anomalous item. This is a special case of general multivariate Gaussian distribution. However, the modeling approach is computationally cheaper and is easy to understand, which is important in managing test development projects.

DATA SOURCES

We use the elicited features and statistics of the items from historical data of a particular test subject to prepare training, cross validation, and test sets for the modeling. Each field test item has nine attributes or features in the model, which includes features based on both classical test theory and item response theory as follows:

- Proportion of examinees responding in the keyed direction (*P-value*)
- Item-total correlation (point-biserial correlations) for correct option
- *b*-parameter of Rasch model
- Inlier-pattern-sensitive fit statistic (*infit*) and outlier-sensitive fit statistic (*outfit*)
- Mean and 25%, 50%, and 75% percentiles of response time

Each feature represents certain item characteristics. Among the 322 items used in this study, 68 items are proposed for inspection by the reviewers before the posterior review meeting. When a complete vetting is done, 35 are excluded. Based on the final adjudications, we then split the data into three data sets—training, validation, and test sets randomly. The training set contains 194 items (60%). They are unlabeled data for fitting the multivariate Gaussian probability function. In other words, they all are considered good test items. Cross validation and test sets are a mixture of good items and disposed items used to determine the threshold and to evaluate the performance. The validation set has 64 items in total (20%). There are 34 items marked as “review items” in a master inspection list by the reviewers to be discussed in the posterior review. After human inspection, 18 items are finally excluded as flawed items. The test set also has 64 items (20%), where 34 are review items and 17 are flawed items.

The following statements read the data to prepare training, cross validation, and test sets. Labeled sets (`vflag` and `tflag`) are created for determining the threshold. The variable `accept` represents whether the item is a flawed one in the final adjudication of the review meeting. The variable `discuss` indicates whether the item is marked as a “review item”, the potential flawed item to be discussed, before the posterior review.

```
data train;
  input pvalue corr irtb infit outfit tmean tp25 tp50 tp75;
  datalines;
0.3783 0.1792 0.15691 1.11 1.12 153  84 130 201.5
0.1053 0.457  2.00573 0.86 0.61 194 108 168 248
0.2412 0.3559 0.81127 0.96 0.92 150  90 134 189
. . .
;
run;

data valid0;
  input pvalue corr irtb infit outfit tmean tp25 tp50 tp75 accept discuss;
  datalines;
0.6804 0.1728 -1.3504  1.07 1.09  36 15  25  46 0 0
0.4779 0.3627 -0.32104 0.94 0.92 146 80 131 189 0 0
0.2924 0.2528 0.57241  1.05 1.06 168 93 147 219 0 0
. . .
;
run;
```

```
data test0;
  input pvalue corr irtb infit outfit tmean tp25 tp50 tp75 accept discuss;
  datalines;
0.2894 0.3314  0.33784 0.96 0.96   73 31 54    95 0 0
0.7615 0.3381 -1.7807  0.9  0.79   89 48 72   106 0 0
0.4514 0.3368 -0.26498 0.96 0.94 117 57 85.5 137 0 0
. . .
;
run;

*Label sets;
data valid(drop=accept discuss) vflag(keep=accept discuss);
  set valid0;
run;
data test (drop=accept discuss) tflag(keep=accept discuss);
  set test0;
run;
```

DENSITY ESTIMATION

After reading the three data sets (`train`, `valid`, and `test`) and computing the means (`mu`) and variences (`s2`) of the training set, the subroutine `MulGauss` is implemented as a PROC FCMP subroutine for density estimation. We fit a simple Gaussian distribution to estimate the mean and variance vectors of the features. For simplicity's sake, the non-diagonal elements of the covariance matrix are set to zero. Through histogram plots, we do know that three features that relate to test taker's response time are slightly skewed to the right, but they do not substantially deviate from Gaussian. The following annotated code shows how to perform density estimation:

```
*mu and s2;
proc means data=train noprint;
output out=mu (drop=_TYPE_ _FREQ_) mean=;
run;
proc means vardef=n data=train noprint;
output out=s2 (drop=_TYPE_ _FREQ_) var=;
run;

*Multivariate Gaussian distribution;
proc fcmp outlib=work.func.ad;
subroutine MulGauss(fX $, fMu $, fS2 $, fP $);
  *read ds;
  array X[1,1] /nosymbols;  rc1=read_array(fX,  X);
  array mu[1]  /nosymbols;  rc2=read_array(fMu,mu);
  array s2[1]  /nosymbols;  rc3=read_array(fS2,s2);

  *diag s2;
  nc=dim(s2,2);  array ds2[1] /nosymbols;  call dynamic_array(ds2,nc,nc);
  call identity(ds2);
  do i=1 to nc;  ds2[i,i]=s2[1,i];  end;

  *X-mu;
  nr=dim(X,1);
  do i=1 to nr;
    do j=1 to nc;
      X[i,j]=X[i,j]-mu[1,j];
    end;
  end;
  *transpose X-mu, inverse s2, determinant s2;
  array Xt[1] /nosymbols;  call dynamic_array(Xt,nc,nr);
  call transpose(X, Xt);
  array ids2[1] /nosymbols;  call dynamic_array(ids2,nc,nc);
  call inv(ds2, ids2);
  call det(ds2,dds2);
  dds2=dds2**(-0.5);
  cc=(2*constant("pi"))**(-nc/2)*dds2;

  *p;
  array tmp1[1] /nosymbols;  call dynamic_array(tmp1,nr,nc);
  call mult(X,ids2,tmp1);
  array tmp2[1] /nosymbols;  call dynamic_array(tmp2,nr,nc);
  call elemmult(tmp1,X, tmp2);
  array p[1] /nosymbols;  call dynamic_array(p,nr);
  call zeromatrix(p);
  do i=1 to nr;
    do j=1 to nc;
      p[i]=p[i]+tmp2[i,j];
    end;
    p[i]=cc*exp(-0.5*p[i]);
  end;
  rc4=write_array(fP,p,'p');

endsub;
quit;

*Density estimation;
options cmplib=work.func;
data _null_;
  call MulGauss("valid","mu","s2", "pv");
  call MulGauss("test", "mu","s2", "pt");
run;
```

Notable in this subroutine are:

1. The matrix must be declared before use. The arrays are declared with `nosymbols`, which means that they do not include variable names for each array element. Here we only use array subscripting to access the elements. The `read_array` function then dynamically resizes the array to fit the dimensions of the input dataset after the data set is read. With the `dynamic_array` function, we can prepare and resize the declared arrays or matrices using the provided dimensions within a function.
2. The code illustrates the operations of matrix. The corresponding CALL routine functions used in PROC FCMP as follows:

CALL routine	**Description**
IDENTITY	identity matrix
TRANSPOSE	transpose matrix
INV	matrix inverse
DET	matrix determinant
MULT	matrix multiplication
ELEMENT	element-wise multiplication
ZEROMATRIX	create matrix of all zeros

DETERMINING THE THRESHOLD

To determine the best threshold, we compute the *F-measure* by looping through different values of threshold in this experiment. If $p(x)$ falls outside the realm, the item is flagged as anomalous. *F-measure* is a classification measure and is defined as the harmonic mean of *precision* and *recall*:

$$F = 2\frac{precision \times recall}{precision + recall}$$

where

$$presion = \frac{true\ positive}{ture\ positive + false\ positive}$$

$$recall = \frac{true\ positive}{ture\ positive + false\ negative}$$

We first set the step size by creating macro variables for looping through different values. Then in the following PROC FCMP code below, we compute precision and recall, and then *F-measure* to obtain the threshold. Finally, we flag a list of potential flawed items and use `write_array` to write arrays to a data set. There are 17 flawed items among 64 items in the test set. In the original list the human reviewers marked 34 items for further inspection and discussion in the review meeting. The model generates a list of 31 items with *F-measure* 0.593. Comparing the two lists, the model's list contains five extra items and does include eight items in the reviewer's master list. Only one rejected item is not covered by the model. We examine that item closely. It is a rather difficult item for the target test takers, probably because most educators do not cover that content area. The following statements determine the threshold, then flag the anomalous items. The role of validation and test sets can be switched by replacing `vflag` with `tflag`, and `pv` with `pt`. This alternative model also generates a list of 31 items with *F-measure* 0.566. There are two out of the 18 flawed items not in the original list.

```
*Set step size;
proc sql noprint;
  select max(p), min(p), (max(p)-min(p))/1000
  into :xpv, :npv, :spv
  from pv;
quit;
proc sql noprint;
  select max(p), min(p), (max(p)-min(p))/1000
  into :xpt, :npt, :spt
  from pt;
quit;

*Select threshold;
proc fcmp;
  array pval[1] /nosymbols;  rc1=read_array("pv",pval);
  array val[1]  /nosymbols;  rc2=read_array("vflag",val);
  n=dim(pval);  array pred[1] /nosymbols;  call dynamic_array(pred,n);

  do thred=&npv to &xpv by &spv;
    do i=1 to n;
      if pval[i]<thred then pred[i]=1;
      else pred[i]=0;
    end;

   *true positive tp, false positive fp, false negative fn;
    tp=0; fp=0; fn=0;
    do i=1 to n;
      if      pred[i]=1 and val[i,1]=1 then tp=tp+1;
     else if pred[i]=1 and val[i,1]=0 then fp=fp+1;
```

```
        else if pred[i]=0 and val[i,1]=1 then fn=fn+1;
      end;

      *F measure and threshold;
      if (tp+fp)~=0 and (tp+fn)~=0 then do;
        prec=tp/(tp+fp);  rec =tp/(tp+fn);
        F=2*prec*rec/(prec+rec);
        if F > bestF then do;
          bestF=F;
          Threshold=thred;
        end;
      end;
    end;

    put BestF=;
    put Threshold=;

    *flag anomalous items;
    array ptst[1] /nosymbols;  rc3=read_array("pt",ptst);
    m=dim(ptst);
    array outLst[1] /nosymbols;  call dynamic_array(outLst,m,2);
    do i=1 to m;
        outLst[i,1]=i;
      if ptst[i] < Threshold then do;
        put i=;
        outLst[i,2]=1;
      end;
      else
        outLst[i,2]=0;
    end;
    rc4=write_array('outFile',outLst,'item','flag');
  quit;
```

CONCLUSION

PROC FCMP provides an option for users to create custom functions and subroutines like any other SAS function. Since it is part of SAS/BASE®, there is no need to purchase an extra license. This paper provides a glimpse of the flexibility of the FCMP procedure. We demonstrate an anomaly detection method based on multivariate Gaussian distribution to show the use of the FCMP procedure's matrix operation CALL routines. There are a vast of other capabilities and powerful CALL routines other than matrix operations, such as functions for calling SAS code from within functions and several others for special purposes, which may be appreciated by analytics programmers.

REFERENCES

Powers, D. M. W. 2011. “Informedness, Markedness & Correlation.” *Journal of Machine Learning Technologies*, 2:37–63.

Schmeiser, B. C. & Welch, C. J. 2006. In R. L. Brennan (Ed.). *Educational Measurement*. 4th ed. (307–353). Westport, CT: Praeger.

CONTACT INFORMATION

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

Tsung-hsun Tsai
Research League, LLC
ttsai@researchleague.org