SAS/IML Avancé

Programmation & Logiciels Statistiques

Cours 11

Salim Lardjane - Université de Bretagne-Sud

Statistique de base avec IML

On va voir comment utiliser SAS/IML pour :

- Transformer des variables
- Identifier des observations qui vérifient un critère
- Traiter les données manquantes
- Analyser les observations par niveaux d'une variable catégorielle

Transformation de variables

- Il est fréquent de vouloir *transformer* des variables en analyse des données.
- Par exemple, on peut souhaiter *centrer et réduire* une variable.
- Si la distribution d'un variable est très asymétrique, on peut souhaiter lui appliquer une *transformation logarithmique*.
- Cela peut être fait facilement avec SAS/IML.

Transformation de variables

- En plus des transformations logarithmiques, les transformations les plus fréquemment utilisées sont la racine carrée, l'inverse, les puissances et l'exponentielle.
- Toutes sont facilement programmable sous SAS/IML.
- Exemple :

Transformation de variables

```
use Sasuser.Movies;
read all var {“Budget”};
close Sasuser.Movies;

logBudget = log(Budget);
```

Transformation de variables

- La transformation log est bien définie dans le cas précédent car la variable Budget est toujours positive.
- Si un vecteur de données y contient des valeurs négatives, il est usuel de rajouter une constante à toutes les valeurs pour les rendre positives.
- Exemple :

Transformation de variables

```
y = {10   0   -29   -20   273   70};
c = 1-min(y);
log10y = log10(y+c);
print log10y;
```

Identifier des observations

- Il est fréquent en analyse des données de vouloir identifier les observations qui vérifient un certain critère.
- Sous IML, on utilise à cet effet la fonction LOC
- Elle est utilisée en conjonction avec les opérateurs de comparaison
- Exemple :

Identifier des observations

```
x = 1:5;
t = (x > 2);
```

t prend la valeur { 0 0 1 1 1}.

La matrice t est non nulle exactement pour les éléments de x qui vérifient la condition.

On peut alors utiliser la fonction LOC.

Identifier des observations

```
idx = loc(t);
print t, idx;
```

- idx contient a pour valeur {3 4 5}.
- En pratique, il n'est pas nécessaire de créer le vecteur t.
- On aurait pu soumettre directement :

Identifier des observations

```
x = 1:5;
idx = loc(t>2);
```

- Autre exemple :

```
varNames = {“Make”  “Model”
“Mpg_City”  “Engine_Cylinders” };
use Sasuser.Vehicles;
read all var varNames;
```

Identifier des observations

```
idx = loc(Mpg_City >= 33);
Print (Make[idx])
      (Model[idx])
      (Mpg_City[idx]) [label =
"Mpg_City" ];
```

- On peut également soumettre des conditions plus complexes :

Identifier des observations

```
idx = loc(Engine_Cylinders = 6 & Mpg_City > 30);
numSatisfy = ncol(idx);
print numSatisfy  “véhicules vérifient le critère.” ;
print (Make[idx]) (Model[idx]) (Mpg_City[idx]) [label =  “Mpg_City”  ];
```

Identifier des observations

- *On peut ainsi utiliser la fonction LOC pour :*
- Identifier des valeurs extrêmes dans le cas univarié
- Identifier des observations ou variables qui contiennent des valeurs manquantes
- Identifier les observations qui sont à plus ou moins d'une certaine distance par rapport à la moyenne ou à la médiane

Identifier des observations

- Détecter des valeurs aberrantes dans un modèle de régression en identifiant les observations pour lesquels les résidus absolus sont grands
- Détecter les observations qui ont une grande influence sur la régression en identifier celles pour lesquelles la valeur du D de Cook, du levier ou du PRESS est élevée.

Identifier des observations

- Nous allons voir à présent comment détecter des outliers dans un modèle de régression :

```
varNames = {“Make”  “Model”  “Mpg_City”
“Mpg_Hwy”  “Hybrid” };
use Sasuser.Vehicles;
read all var varNames;
Close Sasuser.Vehicles;
```

Identifier des observations

```
Pred_Mpg_Hwy = 0.24 + 1.35*Mpg_City;
Residual = Mpg_Hwy - Pred_Mpg_Hwy;
Idx = loc(abs(residual) > 10);
print (Make[idx])
      (Model[idx])
      (redidual[idx]) [label = "Residual" ]
      (Hybrid[idx]) [label = "Hybrid ?" ];
```

Identifier des observations

- Comment traiter le cas où aucune observation ne vérifie les critères ?
- Dans ce cas, LOC renvoie une matrice vide, qui ne peut être utilisée comme index (erreur).
- Exemple de solution :

```
if (ncol(idx) > 0) then
     print …
else
     print "Aucune observation ne vérifie… ;
```

Affecter des valeurs à des observations qui vérifient un critère

- Exemple :

```
varNames = { “Mpg_City”    “Mpg_Hwy”
“Hybrid” };
use Sasuser.Vehicles;
read all var varNames;
Close Sasuser.Vehicles;

Pred = j(nrow(Hybrid),1);
Idx = loc(Hybrid = 1);
```

Affecter des valeurs à des observations qui vérifient un critère

```
if ncol(idx) > 0 then
     Pred[idx] = 6.91 + 0.7 * Mpg_City[idx];
idx = loc(Hybrid ^= 1);
if ncol(idx) > 0 then
     Pred[idx] = 0.09 + 1.36*Mpg_City[idx];
```

- Les valeurs prédites pour les véhicules hybrides sont basées sur un modèle; les valeurs prédites pour les véhicules traditionnels sur un autre.

Affecter des valeurs à des observations qui vérifient un critère

- On aurait pu également le programmer de la façon suivante :

```
Pred = choose(Hybrid = 1,
              6.91 + 0.7 * Mpg_City
              0.09+1.36*Mpg_City);
```

Traiter les valeurs manquantes

- Il est fréquent d'avoir des valeurs manquantes dans les données.
- Il existe *deux façons* de traiter les valeurs manquantes lors de l'analyse des données : *supprimer* toutes les observations pour lesquelles au moins une valeur est manquante ou *imputer* des valeurs pour remplacer les valeurs manquantes.

Traiter les valeurs manquantes

- La plupart des procédures SAS (par exemple, UNIVARIATE, REG et PRINCOMP) suppriment les observations pour lesquelles certaines valeurs sont manquantes.
- MI et MIANALYZE (SAS/STAT) utilisent des techniques *d'imputation multiple*, ce qui ne sera pas abordé ici.

Traiter les valeurs manquantes

- De nombreuses sous-routines et opérateurs matriciels de SAS/IML ignorent simplement les valeurs manquantes.
- C'est par exemple le cas de la fonction SUM, qui permet d'obtenir la somme des éléments d'une matrice.
- De même, le module Median ne prend pas en compte les valeurs manquantes lors du calcul de la médiane de chaque colonne d'une matrice.

Traiter les valeurs manquantes

- Toutefois, la multiplication matricielle et vectorielle renvoient un message d'erreur si certaines des valeurs sont manquantes.
- C'est le cas également de certaines fonctions SAS/IML.
- Par conséquent, écrire des programmes robustes et réutilisables requiert de traiter explicitement le cas de valeurs manquantes.

Traiter les valeurs manquantes

- Exemple *à première vue* correct :

```
use Sasuser.Movies;
read all var {“World_Gross”};
close Sasuser.Movies;

wrong_mean =
sum(World_Gross)/nrow(World_Gross);
Print wrong_mean;
```

(résultat : 116.76681)

Traiter les valeurs manquantes

- La valeur de la moyenne obtenue à l'aide du programme ci-dessus est *fausse.*
- En effet, SUM ignorent les valeurs manquantes, alors que NROW compte *toutes* les lignes, même celles pour lesquelles les valeurs sont manquantes.
- Pour résoudre le problème, on peut soit utiliser des fonctions ignorant les valeurs manquantes, soit les supprimer explicitement.

Traiter les valeurs manquantes

- Exemple correct (première version) :

```
mean1 = World_Gross[:];
mean2 = mean(World_Gross);
mean3 =
sum(World_Gross)/countn(World_Gross);
Print mean1 mean2 mean3;
```

Résultat : 125.50684 125.50684 125.50684

Traiter les valeurs manquantes

- Exemple correct (deuxième version) :

```
nonMissing = loc(World_Gross ^= .);
If ncol(nonMissing) = 0 then mean = .;
else do;
      World_Gross =  World_Gross[nonMissing];
      mean4 = sum(World_Gross)/nrow(World_Gross);
end;
print mean4;
```

- Résultat : 125.50684

Analyses par catégories

- La fonction LOC est souvent utilisée en conjonction avec la fonction UNIQUE lorsqu'on analyse des variables catégorielles.
- La fonction UNIQUE renvoie un vecteur qui contient une liste triée de valeurs uniques.
- Ces deux fonctions renvoient toutes deux un vecteur ligne.
- On peut donc utiliser NCOL pour compter le nombre de valeurs uniques.

Analyses par catégories

- On peut également utiliser NCOL pour compter le nombre d'observations associées à chaque valeur unique.
- Exemple :

```
use Sasuser.Movies;
read all var { "MPAARating" };
close Sasuser.Movies;
```

Analyses par catégories

```
categories = unique(MPAARating);
count = j(ncol(categories),1,0);
do i = 1 to ncol(categories);
     idx = loc(MPAARating = categories[i]);
     count[i] = ncol[idx];
end;
print count[rowname = categories];
```

- Cette technique peut également être utilisée de la façon suivante :

Analyses par catégories

- Si c est une variable catégorielle, alors les instructions suivantes permettent d'analyser les observations dans chaque catégorie de c séparément :

```
uC = unique(c);
do i = 1 to ncol(uC);
      idx = loc(c = uC[i]);
      /* Analyser les observations pour idx */
end;
```

Analyses par catégories

- On peut également utiliser la technique précédente pour obtenir un tableau croisé.
- Exemple :

```
use Sasuser.Movies;
read all var { “Year”  “MPAARating” };
close Sasuser.Movies;
```

Analyses par catégories

```
uYear = unique(Year);
uMPAARating = unique(MPAARating);
Table = j(ncol(uYear),ncol(uMPAARating),0);
do j = 1 to ncol(uMPAARating);
      jdx = loc(MPAARating=uMPAARating[j]);
      t = Year[jdx];
      do i = 1 to ncol(uYear);
            idx = loc(t=uYear[i]);
            Table[i,j] = ncol[idx];
      end;
end;
YearLabel = char(uYear,4);
print Table[rowname=YearLabel colname=uMPAARating];
```

Modules SAS/IML

- Il est possible sous SAS/IML de définir ses propres *modules*.
- Un module est une fonction ou sous-routine *définie par l'utilisateur* qui peut être appelée à partir d'un programme.
- Un module qui renvoie une valeur est appelé un *module de fonction*.
- Un module qui ne renvoie pas de valeur est appelé *module de sous-routine*.

Modules SAS/IML

- Un module est défini par les instructions START et FINISH.
- L'instruction START défini le nom du module et ses arguments.
- L'instruction RETURN est utilisée pour renvoyer une valeur à partir d'un module de fonction.
- Exemple : *module calculant le coefficient de corrélation de Pearson*

Modules SAS/IML

```
start CorrCoef(x,y);
     xStd = standard(x);
     yStd = standard(y);
     df = nrow(x)-1;
     return(t(xStd)*yStd/df);
finish;
u = {1,2,3,4,5};
v = {2,3,1,4,4};
r = CorrCoef(u,v);
print r;
```

Variables locales

- Un module contient en général son propre ensemble de variables.
- Cela signifie que les noms des variables utilisées dans le module n'entrent pas en conflit avec les mêmes noms de variables utilisés en dehors du module.
- Si SHOW NAMES est utilisé *à l'intérieur* d'un module, il affiche les noms, dimensions et types des matrices *propres au module*.

Symboles globaux

- On peut déclarer une variable comme globale en utilisant la clause optionnelle GLOBAL dans l'instruction START.
- Exemple :

```
start HasValue(x) global(g_Value);
      idx = loc(x=g_Value);
      return(ncol(idx) > 0);
finish;
```

Symboles globaux

```
g_Value = 1;
v = {4,2,1,3,8};
z = HasValue(v);
```

Arguments par référence

- SAS/IML passe les matrices en arguments des modules *par référence*, ce qui signifie que si un argument est modifié *à l'intérieur* de module, il est aussi modifié *à l'extérieur* du module.
- Exemple :

Arguments par référence

```
start ReverseRows(x);
      r = x[nrow(x):1, ];
      x = r;
finish;

u = {1,2,3,4,5};
run ReverseRows(u);
print u;
```

Evaluation des arguments

- Le langage SAS/IML résout toutes les *expressions matricielles* et les stocke dans des matrices *temporaires* avant d'appeler un module.
- Exemple :

```
w = {1 2, 2 3, 3 1, 4 4, 5 4};
run ReverseRows(w[,1]);
print w;
```

- w est inchangé.

Sauvegarde des modules

- Un avantage important des modules est qu'il est possible de les sauvegarder puis de les appeler à partir d'un nouveau programme.
- C'est fait à l'aide de l'instruction STORE.
- Pour charger un module sauvegardé, on utilise l'instruction LOAD.
- Exemple :

Sauvegarde des modules

```
start LocNonMissingRows(x);
      c = cmiss(x);
      r = c[,+];
      nonMissingIdx = loc(r=0);
      return(nonMissingIdx);
finish;
store module = LocNonMissingRows;
```

Sauvegarde des modules

- Les modules sont sauvegardés par défaut dans un sous-répertoire du répertoire Work.
- Si on souhaite les sauvegarder de façon permanente, on spécifie un autre répertoire à l'aide de l'instruction RESET STORAGE.
- Cette instruction doit également être utilisée avant un LOAD pour charger un module sauvegardé de façon permanente.

IMLMLIB

- SAS/IML est distribué avec un ensemble de modules prédéfinis, appelés les modules IMLMLIB.
- Il sont stockés dans un catalogue SAS nommé IMLMLIB.
- Il est possible de les appeler sans effectuer de LOAD.
- Exemples : Corr, Median, Quartiles, Standard…

Références

- *Statistical Programming with SAS/IML Software*, Rick Wicklin, *SAS Publishing*.