Title

stata.com

cluster programming subroutines — Add cluster-analysis routines

Description Remarks and examples Reference Also see

Description

This entry describes how to extend Stata's `cluster` command; see [MV] **cluster**. Programmers can add subcommands to `cluster`, add functions to `cluster generate` (see [MV] **cluster generate**), add stopping rules to `cluster stop` (see [MV] **cluster stop**), and set up an alternative command to be executed when `cluster dendrogram` is called (see [MV] **cluster dendrogram**).

The `cluster` command also provides utilities for programmers; see [MV] **cluster programming utilities** to learn more.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Adding a cluster subroutine

You add a `cluster` subroutine by creating a Stata program with the name `cluster_`*subcmdname*. For example, to add the subcommand `xyz` to `cluster`, create `cluster_xyz.ado`. Users could then execute the `xyz` subcommand with

```
cluster xyz ...
```

Everything entered on the command line after `cluster xyz` is passed to the `cluster_xyz` command.

You can add new clustering methods, new cluster-management tools, and new postclustering programs. The `cluster` command has subcommands that can be helpful to cluster-analysis programmers; see [MV] **cluster programming utilities**.

▷ Example 1

We will add a `cluster` subroutine by writing a simple postcluster-analysis routine that provides a cross-tabulation of two cluster-analysis grouping variables. The syntax of the new command will be

`cluster mycrosstab` *clname1 clname2* [`,` *tabulate_options*]

Here is the program:

```
program cluster_mycrosstab
        version 17.0
        gettoken clname1 0 : 0 , parse(" ,")
        gettoken clname2 rest : 0 , parse(" ,")
        cluster query `clname1'
        local groupvar1 `r(groupvar)'
        cluster query `clname2'
        local groupvar2 `r(groupvar)'
        tabulate `groupvar1' `groupvar2' `rest'
end
```

See [P] **gettoken** for information on the `gettoken` command, and see [R] **tabulate twoway** for information on the `tabulate` command. The `cluster query` command is one of the cluster programming utilities that is documented in [MV] **cluster programming utilities**.

We can demonstrate `cluster mycrosstab` in action. This example starts with two cluster analyses, `cl1` and `cl2`. The dissimilarity measure and the variables included in the two cluster analyses differ. We want to see how closely the two cluster analyses match.

```
. use https://www.stata-press.com/data/r17/auto
(1978 automobile data)
. cluster kmeans gear head tr, L1 k(5) name(cl1) start(krandom(55234))
> gen(cl1gvar)
. cluster kmeans tr tu mpg, L(1.5) k(5) name(cl2) start(krandom(22132))
> gen(gvar2)
. cluster list, type method dissim var
cl2  (type: partition,  method: kmeans,  dissimilarity: L(1.5))
      vars: gvar2 (group variable)
cl1  (type: partition,  method: kmeans,  dissimilarity: L1)
      vars: cl1gvar (group variable)
. cluster mycrosstab cl1 cl2, chi2
```

	Cluster ID					
Cluster ID	1	2	3	4	5	Total
1	10	7	0	0	4	21
2	10	0	0	0	0	10
3	0	0	4	5	2	11
4	0	1	6	4	8	19
5	0	11	1	0	1	13
Total	20	19	11	9	15	74

```
Pearson chi2(16) =  97.3723   Pr = 0.000
```

The `chi2` option was included to demonstrate that we were able to exploit the existing options of `tabulate` with little programming effort. We just pass along to `tabulate` any of the extra arguments received by `cluster_mycrosstab`.

◁

Adding a cluster generate function

Programmers can add functions to the `cluster generate` command (see [MV] **cluster generate**) by creating a command called `clusgen_`*name*. For example, to add a function called `abc()` to `cluster generate`, you could create `clusgen_abc.ado`. Users could then execute

`cluster generate` *newvar* `= abc( ... ) ...`

Everything entered on the command line following `cluster generate` is passed to `clusgen_abc`.

▷ Example 2

Here is the beginning of a `clusgen_abc` program that expects an integer argument and has one option called `name(`*clname*`)`, which gives the name of the cluster. If `name()` is not specified, the name defaults to that of the most recently performed cluster analysis. We will assume, for illustration purposes, that the cluster analysis must be `hierarchical` and will check for this in the `clusgen_abc` program.

```
program clusgen_abc
        version 17.0
        // we use gettoken to work our way through the parsing
        gettoken newvar 0 : 0 , parse(" =")
        gettoken temp 0 : 0 , parse(" =")
        if `"`temp'"' != "=" {
                error 198
        }
        gettoken temp 0 : 0 , parse(" (")
        if `"`temp'"' != "abc" {
                error 198
        }
        gettoken funcarg 0 : 0 , parse(" (") match(temp)
        if `"`temp'"' != "(" {
                error 198
        }
        // funcarg holds the integer argument to abc()
        confirm integer number `funcarg'
        // we can now use syntax to parse the option
        syntax [, Name(str) ]
        // cluster query will give us the list of cluster names
        if `"`name'"' == "" {
                cluster query
                local clnames `r(names)'
                if "`clnames'" == "" {
                        di as err "no cluster solutions defined"
                        exit 198
                }
                // first name in the list is the latest clustering
                local name : word 1 of `clnames'
        }
        // cluster query followed by name will tell us the type
        cluster query `name'
        if "`r(type)'" != "hierarchical" {
                di as err "only allowed with hierarchical clustering"
                exit 198
        }
        /*
           you would now pull more information from the call of
                       cluster query `name'
           and do your computations and generate `newvar'
        */
        ...
end
```

See [MV] **cluster programming utilities** for details on the `cluster query` command.

◁

Adding a cluster stopping rule

Programmers can add stopping rules to the `rule()` option of the `cluster stop` command (see [MV] **cluster stop**) by creating a Stata program with the name `clstop_`*name*. For example, to add a stopping rule named `mystop` so that `cluster stop` would now have a `rule(mystop)` option, you could create `clstop_mystop.ado` defining the `clstop_mystop` program. Users could then execute

`cluster stop` [*clname*]`, rule(mystop)` ...

The `clstop_mystop` program is passed the cluster name (*clname*) provided by the user (or the name of the current cluster result if no name is specified), followed by a comma and all the options entered by the user except for the `rule(mystop)` option.

▷ Example 3

We will add a `rule(stepsize)` option to `cluster stop`. This option implements the simple step-size stopping rule (see Milligan and Cooper 1985), which computes the difference in fusion values between levels in a hierarchical cluster analysis. (A fusion value is the similarity or dissimilarity measure at which clusters are fused or split in the hierarchical cluster structure.) Large values of the step-size stopping rule indicate groupings with more distinct cluster structure.

Examining cluster dendrograms (see [MV] **cluster dendrogram**) to visually determine the number of clusters is equivalent to using a visual approximation to the step-size stopping rule.

Here is the `clstop_stepsize` program:

```
program clstop_stepsize, sortpreserve rclass
        version 17.0
        syntax anything(name=clname) [, Depth(integer -1) ]
        cluster query `clname'
        if "`r(type)'" != "hierarchical" {
                di as error ///
                    "rule(stepsize) only allowed with hierarchical clustering"
                exit 198
        }
        if "`r(pseudo_heightvar)'" != "" {
                di as error "dendrogram reversals encountered"
                exit 198
        }
        local hgtvar `r(heightvar)'
        if `""`r(similarity)'"' != "" {
                sort `hgtvar'
                local negsign "-"
        }
        else if `""`r(dissimilarity)'"' != "" {
                gsort -`hgtvar'
        }
        else {
                di as error "dissimilarity or similarity not set"
                exit 198
        }
        quietly count if !missing(`hgtvar')
        local depth = cond(`depth'<=1, r(N), min(`depth',r(N)))
        tempvar diff
        qui gen double `diff'=`negsign'(`hgtvar'-`hgtvar'[_n+1]) if _n<`depth'
        di
        di as txt "Depth" _col(10) "Stepsize"
        di as txt "{hline 17}"
        forvalues i = 1/`= `depth'-1' {
                local j = `i' + 1
                di as res `j' _col(10) %8.0g `diff'[`i']
                return scalar stepsize_`j' = `diff'[`i']
        }
        return local rule "stepsize"
end
```

See [P] **syntax** for information about the `syntax` command, [P] **forvalues** for information about the `forvalues` looping command, and [P] **macro** for information about the `'= ... '` macro function. The `cluster query` command is one of the cluster programming utilities that is documented in [MV] **cluster programming utilities**.

With this program, users can obtain the step-size stopping rule. We demonstrate this process by using an average-linkage hierarchical cluster analysis on the data found in the second example of [MV] **cluster linkage**. The dataset contains 30 observations on 60 binary variables. The simple matching coefficient is used as the similarity measure in the average-linkage clustering.

```
. use https://www.stata-press.com/data/r17/homework, clear
. cluster a a1-a60, measure(match) name(alink)
. cluster stop alink, rule(stepsize) depth(15)
```

Depth	Stepsize
2	.065167
3	.187333
4	.00625
5	.007639
6	.002778
7	.005952
8	.002381
9	.008333
10	.005556
11	.002778
12	0
13	0
14	.006667
15	.01

In the `clstop_stepsize` program, we included a `depth()` option. `cluster stop`, when called with the new `rule(stepsize)` option, can also have the `depth()` option. Here we specified that it stop at a depth of 15.

The largest step size, .187, happens at the three-group level of the hierarchy. This number, .187, represents the difference between the matching coefficient created when two groups are formed and that created when three groups are formed in this hierarchical cluster analysis.

The `clstop_stepsize` program could be enhanced by using a better output table format. An option could also be added that stores the results in a matrix.

◁

Applying an alternate cluster dendrogram routine

Programmers can change the behavior of the `cluster dendrogram` command (alias `cluster tree`); see [MV] **cluster dendrogram**. This task is accomplished by using the `other()` option of the `cluster set` command (see [MV] **cluster programming utilities**) with a *tag* of `treeprogram` and with *text* giving the name of the command to be used in place of the standard Stata program for `cluster dendrogram`. For example, if you had created a new hierarchical cluster-analysis method for Stata that needed a different algorithm for producing dendrograms, you would use the command

`cluster set` *clname*`, other(treeprogram` *progname*`)`

to set *progname* as the program to be executed when `cluster dendrogram` is called.

▷ Example 4

If we were creating a new hierarchical cluster-analysis method called `myclus`, we could create a program called `cluster_myclus` (see *Adding a cluster subroutine*). If `myclus` needed a different dendrogram routine from the standard one used within Stata, we could include the following line in `cluster_myclus.ado` at the point where we set the cluster attributes.

```
cluster set `clname', other(treeprogram myclustree)
```

We could then create a program called `myclustree` in a file called `myclustree.ado` that implements the particular dendrogram program needed by `myclus`.

◁

Reference

Milligan, G. W., and M. C. Cooper. 1985. An examination of procedures for determining the number of clusters in a dataset. *Psychometrika* 50: 159–179. https://doi.org/10.1007/BF02294245.

Also see

[MV] **cluster** — Introduction to cluster-analysis commands

[MV] **clustermat** — Introduction to clustermat commands

[MV] **cluster programming utilities** — Cluster-analysis programming utilities