Split-Apply-Combine with Dynamic Grouping

Mark van der Loo

Statistics Netherlands and Leiden University

2024-07-11

Contents

- Hierarchy and aggregation
- Dynamic grouping
- The `accumulate` R package

Split-Apply-Combine

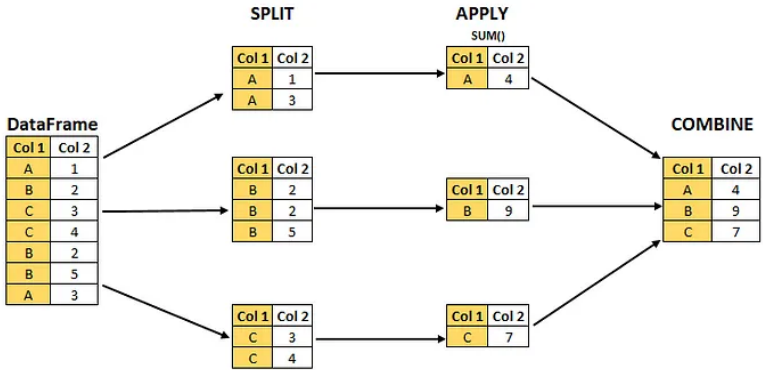
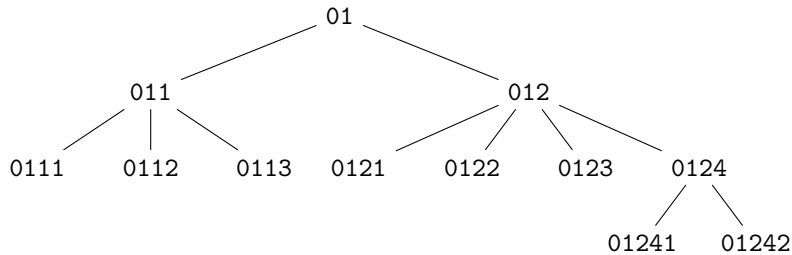


Image credit: Anurag Pandey

Dynamic grouping by NACE code



- Report the mean by the 4-digit code. If there are less than 5 records, compute the mean by using all records one level higher. Recurse until 5 records or stop and return `NA`.

Dynamic Grouping with Multiple variables

- Report the mean by size class $\times$ 4-digit NACE
- If there are less than 5 records (collapsing scheme)
  1. Drop size class
  2. Collapse to 3-digit NACE
  3. Collapse to 2-digit NACE
  4. Otherwise return `NA`.

The Accumulate package

```
library("accumulate")
```

Example data

	sbi	size	industrial	trade	other	other_income	total
1	3410	8	151722	2135	0	-1775	152082
2	2840	7	50816	NA	158	949	59876
3	2752	5	4336	NA	0	36	4959
4	3120	6	18508	NA	0	80	20682
5	2524	7	21071	0	0	442	21513
6	3410	6	24220	1069	0	239	25528

Interfaces

To specify:

what	How
Collapsing scheme	`formula` or `data.frame`
Collapse criterion	`function`
Data to aggregate	`base`- or `dplyr`-like

Interfaces

To specify:

what	How
Collapsing scheme	`formula` or `data.frame`
Collapse criterion	`function`
Data to aggregate	`base`- or `dplyr`-like

`accumulate()`
Aggregate over all non-grouping variables. Just like `base::aggregate()`.

`cumulate()`
Specify output colums. Just like `dplyr::summarize()`

accumulate()

```
a <- accumulate(producers
                , collapse = sbi*size ~ sbi
                , test     = function(d) nrow(d) >= 5
                , fun      = mean, na.rm=TRUE)
head(round(a), 3)
```

```
   sbi size level industrial trade other other_income  total
1 3410    8     1     364397  2859    33          353 546117
2 2840    7     0      23160   823    49          329  25812
3 2752    5    NA         NA    NA    NA           NA     NA
```

accumulate()

```
producers$sbi3 <- substr(producers$sbi,1,3)
a <- accumulate(producers
                , collapse = sbi*size ~ sbi + sbi3
                , test     = min_records(5)
                , fun      = mean, na.rm=TRUE)
head(round(a), 3)
```

```
   sbi size level industrial trade other other_income  total
1 3410    8     1     364397  2859    33          353 546117
2 2840    7     0      23160   823    49          329  25812
3 2752    5     2      19526    39    52          151  20603
```

cumulate()

```
a <- cumulate(producers
              , collapse     = sbi*size ~ sbi + sbi3
              , test         = min_records(5)
              , industrial   = median(industrial, na.rm=TRUE)
              , other_income = mean(other_income, na.rm=TRUE))
head(round(a), 3)
```

```
   sbi size level industrial other_income
1 3410    8     1      97395          353
2 2840    7     0      17729          329
3 2752    5     2       9540          151
```

Support for complex output

```
a <- cumulate(producers
           , collapse     = sbi*size ~ sbi + sbi3
           , test         = min_records(5)
           , model        = lm(total ~ industrial ) )
head(a, 3)
```

```
   sbi size level model
1 3410    8     1  <lm>
2 2840    7     0  <lm>
3 2752    5     2  <lm>
```

```
a$model[[1]]
```

```

Call:
lm(formula = total ~ industrial)

Coefficients:
(Intercept)   industrial
  40646.401        1.387
```

Facilities

Derive collapsing scheme from hierarchical classification

```
csh_from_digits(c("123","124"),levels=1)
```

```
   A0 A1
1 123 12
2 124 12
```

Facilities

Derive collapsing scheme from hierarchical classification

```
csh_from_digits(c("123","124"),levels=1)
```

```
   A0 A1
1 123 12
2 124 12
```

Test functions

`min_records()`, `frac_complete()`, `min_complete()`

```
# use validation rules for testing
library(validate)
rules <- validator(var(x, na.rm=TRUE)<2, nrow(.) >= 5)
test <- from_validator(rules)
```

Facilities

Derive collapsing scheme from hierarchical classification

```
csh_from_digits(c("123","124"),levels=1)
```

```
   A0 A1
1 123 12
2 124 12
```

Test functions

`min_records()`, `frac_complete()`, `min_complete()`

```
# use validation rules for testing
library(validate)
rules <- validator(var(x, na.rm=TRUE)<2, nrow(.) >= 5)
test <- from_validator(rules)
```

Test your `test()` function

```
smoke_test(data, test)
```

Analyses

Computational complexity

Operation	Complexity
SAC	$\mathcal{O}(m)$
SACCG	$\mathcal{O}(nm)$

Where m is the number of groups,
n the number of collapse levels.

Algorithm 2: Split-Apply-Combine with Collapsing Groups: SACCG(U, ϕ, β, C)

Input : A finite set U, an aggregator $\phi : 2^U \to X$, a test function $\beta : 2^U \to \mathbb{B}$, and a collapsing sequence $C \equiv U \xrightarrow{f_0} A \xrightarrow{f_1} A_1 \xrightarrow{f_2} \dots \xrightarrow{f_n} A_n$.

Output: R: the value of ϕ for every part of U, for which a suitable collapsing group can be found, as a set of triples $(a, k, x) \in A \times \underline{n} \times X$ where $\underline{n} = \{0, 1, \dots, n\}$.

```
R = {};
for a ∈ A do
    i = 0 ;                                   // Initiate collapse level
    d = f*({a}) ;                                   // Get subset of U
    while i < n ∧ ¬β(d) do
        i = i + 1 ;                           // Increase collapse level
        d = (f* ∘ F_i* ∘ F_i)(a) ;             // Collapse and get subset
    end
    if i < n ∨ β(d) then
        R = R ∪ {(a, i, φ(d))};
    end
end
```

Observation

Split-Apply-Combine is a formal special case of Split-Apply-Combine with Collapsing Groups. Setting the test function to `TRUE` reduces `SACCG` to `SAC`.

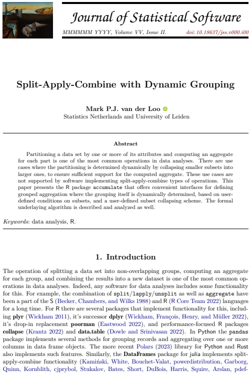
Summary

Results

- Formal algorithm for split-apply-combine with collapsing groups
- Implemented in `accumulate` R package (on CRAN)
- Collapsing scheme specifyable via
  - formula (based on variables in data)
  - data frame (user-defined, external collapsing scheme)
- Flexible testing of subgroups
  - Built-in functions
  - Custom function
  - Derive from `validate` rules

Thank you

markvanderloo.eu/publications.html



Journal of Statistical Software
MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

Split-Apply-Combine with Dynamic Grouping

Mark P.J. van der Loo
Statistics Netherlands and University of Leiden

Abstract

Partitioning a data set by one or more of its attributes and computing an aggregate for each part is one of the most common operations in data analyses. There are use cases where the partitioning is determined dynamically by collapsing smaller subsets into larger ones, to ensure sufficient support for the computed aggregate. These use cases are not supported by software implementing split-apply-combine types of operations. This paper presents the R package `accumulate` that offers convenient interfaces for defining grouped aggregation where the grouping itself is dynamically determined, based on user-defined conditions on subsets, and a user-defined subset collapsing scheme. The formal underlaying algorithm is described and analyzed as well.

Keywords: data analysis, R.

1. Introduction

The operation of splitting a data set into non-overlapping groups, computing an aggregate for each group, and combining the results into a new dataset is one of the most common operations in data analyses. Indeed, any software for data analyses includes some functionality for this. For example, the combination of `split/lapply/unsplit` as well as `aggregate` have been a part of the S (Becker, Chambers, and Wilks 1988) and R (R Core Team 2022) languages for a long time. For R there are several packages that implement functionality for this, including **plyr** (Wickham 2011), it's successor **dplyr** (Wickham, François, Henry, and Müller 2022), it's drop-in replacement **poorman** (Eastwood 2022), and performance-focused R packages **collapse** (Krantz 2022) and **data.table** (Dowle and Srinivasan 2022). In Python the pandas package implements several methods for grouping records and aggregating over one or more columns in data frame objects. The more recent Polars (2023) library for Python and Rust also implements such features. Similarly, the **DataFrames** package for julia implements split-apply-combine functionality (Kamiński, White, Bouchet-Valat, powerdistribution, Garborg, Quinn, Kornblith, cjprybol, Stukalov, Bates, Short, DuBois, Harris, Squire, Arslan, pdef-

MPJ van der Loo (2024)
The Journal of Statistical Software
(Accepted for publication)