Package ‘fplyr’

June 22, 2020

Type Package

Title Apply Functions to Blocks of Files

Version 1.2.1

Maintainer Federico Marotta <federicomarotta@mail.com>

Description Read and process a large delimited file block by block. A block consists of all the contiguous rows that have the same value in the first field. The result can be returned as a list or a data.table, or even directly printed to an output file.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends data.table (>= 1.1.4)

Imports iotools (>= 0.2.5), parallel (>= 3.5.1)

Suggests testthat, knitr, rmarkdown

URL https://github.com/fmarotta/fplyr

BugReports https://github.com/fmarotta/fplyr/issues

RoxygenNote 7.0.2

VignetteBuilder knitr

NeedsCompilation no

Author Federico Marotta [aut, cre] (<https://orcid.org/0000-0002-0174-3901>)

Repository CRAN

Date/Publication 2020-06-22 18:10:02 UTC

R topics documented:

fdply *Read some chunks from a file into a* `data.table`

Description

This function is useful to quickly glance at a big chunked file. It is similar to the `head()` function, except that it does not read the first few lines, but rather the first few blocks of the file. By default, only the first block will be read; it is not advisable to read a large number of blocks in this way because they may occupy a lot of memory. The blocks are saved to a `data.table`. See `?fplyr` for the definitions of chunked file and block.

Usage

```
fdply(
  input,
  nblocks = 1,
  key.sep = "\t",
  sep = "\t",
  skip = 0,
  colClasses = NULL,
  header = TRUE,
  stringsAsFactors = FALSE,
  select = NULL,
  drop = NULL,
  col.names = NULL,
  parallel = 1
)
```

Arguments

`input`	Path of the input file.
`nblocks`	The number of blocks to read.
`key.sep`	The character that delimits the first field from the rest.
`sep`	The field delimiter (often equal to `key.sep`).
`skip`	Number of lines to skip at the beginning of the file
`colClasses`	Vector or list specifying the class of each field.
`header`	Whether the file has a header.
`stringsAsFactors`	Whether to convert strings into factors.
`select`	The columns (names or numbers) to be read.
`drop`	The columns (names or numbers) not to be read.
`col.names`	Names of the columns.
`parallel`	Number of cores to use.

Value

A `data.table` containing the file truncated to the number of blocks specified.

Slogan

fdply: from **f**ile to **d**ata.table

`ffply` *Read, process each block and write the result*

Description

Suppose you want to process each block of a file and the result is again a `data.table` that you want to print to some output file. One possible approach is to use `l <-flply(...)` followed by `do.call(rbind,l)` and `fwrite`, but this would be slow. `ffply` offers a faster solution to this problem.

Usage

```
ffply(
  input,
  output = "",
  FUN,
  ...,
  key.sep = "\t",
  sep = "\t",
  skip = 0,
  header = TRUE,
  nblocks = Inf,
  stringsAsFactors = FALSE,
  colClasses = NULL,
  select = NULL,
  drop = NULL,
  col.names = NULL,
  parallel = 1
)
```

Arguments

`input`	Path of the input file.
`output`	String containing the path to the output file.
`FUN`	Function to be applied to each block. It must take at least two arguments, the first of which is a `data.table` containing the current block, *without the first field*; the second argument is a character vector containing the value of the first field for the current block.
`...`	Additional arguments to be passed to FUN.

`key.sep` The character that delimits the first field from the rest.

`sep` The field delimiter (often equal to `key.sep`).

`skip` Number of lines to skip at the beginning of the file

`header` Whether the file has a header.

`nblocks` The number of blocks to read.

`stringsAsFactors` Whether to convert strings into factors.

`colClasses` Vector or list specifying the class of each field.

`select` The columns (names or numbers) to be read.

`drop` The columns (names or numbers) not to be read.

`col.names` Names of the columns.

`parallel` Number of cores to use.

Value

Returns the number of chunks that were processed. As a side effect, writes the processed `data.table` to the output file.

Slogan

ffply: from **f**ile to **f**ile

Examples

```
f1 <- system.file("extdata", "dt_iris.csv", package = "fplyr")
f2 <- tempfile()

# Copy the first two blocks from f1 into f2 to obtain a shorter but
# consistent version of the original input file.
ffply(f1, f2, function(d, by) {return(d)}, nblocks = 2)

# Compute the mean of the columns for each species
ffply(f1, f2, function(d, by) d[, lapply(.SD, mean)])

# Reshape the file, block by block
ffply(f1, f2, function(d, by) {
    val <- do.call(c, d)
    var <- rep(names(d), each = nrow(d))
    data.table(Var = var, Val = val)
})
```

flply *Read, process each block and return a list*

Description

With flply() you can apply a function to each block of the file separately. The result of each function is saved into a list and returned. flply() is similar to lapply(), except that it applies the function to each block of the file rather than to each element of a list. It is also similar to by(), except that it does not read the whole file into memory, but each block is processed as soon as it is read from the disk.

Usage

```
flply(
  input,
  FUN,
  ...,
  key.sep = "\t",
  sep = "\t",
  skip = 0,
  header = TRUE,
  nblocks = Inf,
  stringsAsFactors = FALSE,
  colClasses = NULL,
  select = NULL,
  drop = NULL,
  col.names = NULL,
  parallel = 1
)
```

Arguments

input Path of the input file.

FUN A function to be applied to each block. The first argument to the function must be a data.table containing the current block. Additional arguments can be passed with ....

... Additional arguments to be passed to FUN.

key.sep The character that delimits the first field from the rest.

sep The field delimiter (often equal to key.sep).

skip Number of lines to skip at the beginning of the file

header Whether the file has a header.

nblocks The number of blocks to read.

stringsAsFactors Whether to convert strings into factors.

`colClasses`	Vector or list specifying the class of each field.
`select`	The columns (names or numbers) to be read.
`drop`	The columns (names or numbers) not to be read.
`col.names`	Names of the columns.
`parallel`	Number of cores to use.

Value

Returns a list containing, for each chunk, the result of the processing.

Slogan

flply: from **f**ile to **l**ist

Examples

```
f <- system.file("extdata", "dt_iris.csv", package = "fplyr")

# Compute, within each block, the correlation between Sepal.Length and Petal.Length
flply(f, function(d) cor(d$Sepal.Length, d$Petal.Length))

# Summarise each block
flply(f, summary)

# Make a different linear model for each block
block.lm <- function(d) {
  lm(Sepal.Length ~ ., data = d[, !"Species"])
}
lm.list <- flply(f, block.lm)
```

`fmply` *Read, process and write to multiple output files*

Description

Sometimes a file should be processed in many different ways. `fmply()` applies a function to each block of the file; the function should return a list of *m* `data.tables`, each of which is written to a different output file. Optionally, the function can return a list of *m + 1*, where the first *m* elements are `data.tables` and are written to the output files, while the last element is returned as in `flply()`.

Usage

```
fmply(
  input,
  outputs,
  FUN,
  ...,
  key.sep = "\t",
  sep = "\t",
  skip = 0,
  header = TRUE,
  nblocks = Inf,
  stringsAsFactors = FALSE,
  colClasses = NULL,
  select = NULL,
  drop = NULL,
  col.names = NULL,
  parallel = 1
)
```

Arguments

input — Path of the input file.

outputs — Vector of *m* paths for the output files.

FUN — A function to apply to each block. Takes as input a data.table and optionally additional arguments. It should return a list of length *m*, the same length as the outputs vector. The first element of the list is written to the first output file, the second element of the list to the second output file, and so on. Besides these *m* data.tables, it can return an additional element, which is also returned by fmply().

... — Additional arguments to be passed to FUN.

key.sep — The character that delimits the first field from the rest.

sep — The field delimiter (often equal to key.sep).

skip — Number of lines to skip at the beginning of the file

header — Whether the file has a header.

nblocks — The number of blocks to read.

stringsAsFactors — Whether to convert strings into factors.

colClasses — Vector or list specifying the class of each field.

select — The columns (names or numbers) to be read.

drop — The columns (names or numbers) not to be read.

col.names — Names of the columns.

parallel — Number of cores to use.

Value

If FUN returns *m* elements, fmply() returns invisibly the number of blocks parsed. If FUN returns *m + 1* elements, fmply() returns the list of all the last elements. As a side effect, it writes the first *m* outputs of FUN to the outputs files.

Slogan

fmply: from **f**ile to **m**ultiple files

Examples

```
fin <- system.file("extdata", "dt_iris.csv", package = "fplyr")
fout1 <- tempfile()
fout2 <- ""

# Copy the input file to tempfile as it is, and, at the same time, print
# a summary to the console
fmply(fin, c(fout1, fout2), function(d) {
    list(d, data.table(unclass(summary(d))))
})

fout3 <- tempfile()
fout4 <- tempfile()

# Use linear and polynomial regression and print the outputs to two files
fmply(fin, c(fout3, fout4), function(d) {
    lr.fit <- lm(Sepal.Length ~ ., data = d[, !"Species"])
    lr.summ <- data.table(Species = d$Species[1], t(coefficients(lr.fit)))
    pr.fit <- lm(Sepal.Length ~ poly(as.matrix(d[, 3:5]), degree = 3),
                 data = d[, !"Species"])
    pr.summ <- data.table(Species = d$Species[1], t(coefficients(pr.fit)))
    list(lr.summ, pr.summ)
})
```

`fplyr` *fplyr: Read, Process and Write*

Description

This package provides a set of functions to quickly read files chunk by chunk, apply a function to each chunk, and return the result. It is especially useful when the files to be processed don't fit into the RAM. Familiarity with the `data.table` package is essential in order to use `fplyr`.

Definitions

Chunked file: A delimited file where many contiguous rows have the same value on the first field. See the example below.

Block: Any portion of the chunked file such that the first field does not change.

Chunk: Chunks are used internally; they consist of one or more block, but regular users should not be concerned with them, and can consider chunks and blocks as synonyms.

Main functions

The main functions are `ffply` and `flply`. The former writes the processed data into a file, while the latter returns it as a list. The former is also much faster. There is also `fdply`, which returns a `data.table` and is useful to only read a certain number of chunks from the file (one by default). `fmply` is useful when the original file needs to be processed in many ways and each outcome must be written to a different file.

Note

Throughout the documentation of this package, the word ’file’ actually means ’chunked file.’

Examples

A chunked file may look as follows:

V1	V2	V3	V4
ID01	ABC	Berlin	0.1
ID01	DEF	London	0.5
ID01	GHI	Rome	0.3
ID02	ABC	Lisbon	0.2
ID02	DEF	Berlin	0.6
ID02	LMN	Prague	0.8
ID02	OPQ	Dublin	0.7
ID03	DEF	Lisbon	-0.1
ID03	LMN	Berlin	0.01
ID03	XYZ	Prague	0.2

The important thing is that the first field has some contiguous lines that take the same value. The values of the other fields are unimportant. This package is useful to process this kind of files, block by block.

`ftply` *Read, process each block and return a data.table*

Description

`ftply` takes as input the path to a file and a function, and returns a `data.table`. It is a faster equivalent to using `l <-flply(...)` followed by `do.call(rbind,l)`.

Usage

```
ftply(
  input,
  FUN = function(d, by) d,
  ...,
  key.sep = "\t",
  sep = "\t",
  skip = 0,
  header = TRUE,
  nblocks = Inf,
  stringsAsFactors = FALSE,
  colClasses = NULL,
  select = NULL,
  drop = NULL,
  col.names = NULL,
  parallel = 1
)
```

Arguments

`input` Path of the input file.

`FUN` Function to be applied to each block. It must take at least two arguments, the first of which is a `data.table` containing the current block, *without the first field*; the second argument is a character vector containing the value of the first field for the current block.

`...` Additional arguments to be passed to FUN.

`key.sep` The character that delimits the first field from the rest.

`sep` The field delimiter (often equal to `key.sep`).

`skip` Number of lines to skip at the beginning of the file

`header` Whether the file has a header.

`nblocks` The number of blocks to read.

`stringsAsFactors` Whether to convert strings into factors.

`colClasses` Vector or list specifying the class of each field.

`select` The columns (names or numbers) to be read.

`drop` The columns (names or numbers) not to be read.

`col.names` Names of the columns.

`parallel` Number of cores to use.

Details

`ftply` is similar to `ffply`, but while the latter writes to disk the result of the processing after each block, the former keeps the result in memory until all the file has been processed, and then returns the complete `data.table`.

Value

Returns a `data.table` with the results of the processing.

Slogan

ftply: from **f**ile to data.**t**able

Examples

```
f1 <- system.file("extdata", "dt_iris.csv", package = "fplyr")

# Compute the mean of the columns for each species
ftply(f1, function(d, by) d[, lapply(.SD, mean)])

# Read only the first two blocks
ftply(f1, nblocks = 2)
```

Index