Some Additional R Features

Hoang Tran

4/25/2017

R Markdown

This is an R Markdown presentation. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see `http://rmarkdown.rstudio.com`.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Parallel Computing

- What is it?
  - Send jobs to different clusters/processors
  - Parallel vs. serial computing
- When to use it?
  - A task that has to be completed many times, but each task is independent
  - Monte carlo simulations
  - Bootstrap
  - Repeated cross-validation
  - Differential expression of genes/microbiota/etc. (multiple testing)

Parallel Computing in R

- Rule of thumb - anything in an `apply` function can be parallelized
- Best to minimize the number of "dispatches"
- Nested for loop - put outer loop in parallel

Parallel Computing in R

- Requirements
  - A multicore laptop/desktop
  - Parallel packages: `parallel`, `doParallel`, `doMC`, `doSNOW` (required for Windows machines)
- Not recommended to do parallel computation in an IDE (RStudio)
  - In my experience this hasn't been a problem, but it's safer to run parallel code from the console

An OSX/Unix Example

Make a function for generating random data

```
rm(list=ls())
pacman::p_load(parallel)
data_gen <- function(n=100, p=10){
  X <- matrix(rnorm(n*p), nrow=n, ncol=p)
  beta <- matrix(runif(p), ncol=1)
  noise <- matrix(rnorm(n), ncol=1)
  y <- X %*% beta + noise
  mod_fit <- lm(y~X)
  return(median(mod_fit$coefficients))
}
```

An OSX/Unix Example

```
nsim <- 1e3
n <- 100
p <- 10
###serial
system.time(ser_res <- replicate(nsim, data_gen()))
```

```
##    user  system elapsed
##   1.066   0.017   1.095
```

```
###parallel
cl <- makeForkCluster(detectCores())
system.time(par_res <- parSapply(cl, 1:nsim,
                                 function(iter){data_gen()}
```

```
##    user  system elapsed
##   0.003   0.000   0.400
```

```
stopCluster(cl)
```

Steps

- We used `makeForkCluster` from `parallel` to create a backend.
  - Many other ways - `registerDoParallel`, `registerDoMC`
  - Windows - `makeSOCKcluster` and `registerDoSnow`
- We used `detectCores()` to specify the number of cores to use.

```
detectCores()
```

```
## [1] 8
```

Reproducibility

- We didn't set a seed in the previous example
- We can't just write `set.seed(123)` at the beginning of the script - seeds need to be coordinated across nodes
- Use `clusterSetRNGStream`, pass a vector of seeds, or `doRNG` package

Reproducibility

```
###parallel
cl <- makeForkCluster(detectCores())
clusterSetRNGStream(cl, 123)
par_res1 <- parSapply(cl, 1:nsim,
                      function(iter){data_gen()})
clusterSetRNGStream(cl, 123)
par_res2 <- parSapply(cl, 1:nsim,
                      function(iter){data_gen()})
stopCluster(cl)
median(par_res1) - median(par_res2)
```

```
## [1] 0
```

Using `foreach` loops

- It's "cleaner" to use the parallel versions of `apply`, `sapply`, `vapply`, `lapply`, etc.
- Sometimes code will be very involved and it's "easier" to use `foreach` - parallel version of a `for` loop
- Some syntax differences - `foreach` returns a list object by default (can be changed)
- Show html document here

Parallel Computing in Matlab

- Use `parpool` to set up the clusters
- Use `parfor` as the equivalent of a `foreach` loop
- Reproducibility
  - Simulations - set seed at the beginning, pass data sets into the loop
  - Or, generate a vector of integers and use these as the seed inside the loop

Some useful packages in R

- The `tidyverse`: `http://tidyverse.org`
  - Includes - `ggplot2`, `dplyr`, `tibble`, `purrr`, `readr`
- `pacman` - convenient loading/installation of packages
- `knitr` and `rmarkdown` - presentations/documents with inline code (including LaTeX)
  - An inline equation: $N(\mu, \sigma^2)$

The Pipe

- Use %>% for concise manipulation of an object (from `magrittr` but loaded through `tidyverse`)

```
# Good
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mouse) %>%
  bop(on = head)
# Bad
foo_foo <- hop(foo_foo, through = forest)
foo_foo <- scoop(foo_foo, up = field_mice)
foo_foo <- bop(foo_foo, on = head)
```

The Pipe

- Like a "chain" of operations on an object

```
iris_summ <- iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Sepcies = n_distinct(Species)
  )
iris_summ
```

```
## # A tibble: 3 × 4
##      Species Sepal.Length Sepal.Width Sepcies
##       <fctr>        <dbl>       <dbl>   <int>
## 1     setosa        5.006       3.428       1
## 2 versicolor        5.936       2.770       1
## 3  virginica        6.588       2.974       1
```

tibbles

- A modern take on data frames - `https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html`
- Input type is never changed
- Convenient for *list columns* (more later)
- Concise printed output - including data types!
- Rownames are removed - use `rownames_to_column()` to preserve
- Overall - a data frame with best practices

tibbles

```
as_tibble(mtcars)
```

```
## # A tibble: 32 × 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am
## *  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   21.0     6 160.0   110  3.90 2.620 16.46     0     1
## 2   21.0     6 160.0   110  3.90 2.875 17.02     0     1
## 3   22.8     4 108.0    93  3.85 2.320 18.61     1     1
## 4   21.4     6 258.0   110  3.08 3.215 19.44     1     0
## 5   18.7     8 360.0   175  3.15 3.440 17.02     0     0
## 6   18.1     6 225.0   105  2.76 3.460 20.22     1     0
## 7   14.3     8 360.0   245  3.21 3.570 15.84     0     0
## 8   24.4     4 146.7    62  3.69 3.190 20.00     1     0
## 9   22.8     4 140.8    95  3.92 3.150 22.90     1     0
## 10  19.2     6 167.6   123  3.92 3.440 18.30     1     0
## # ... with 22 more rows
```

Default data.frame output

```
mtcars
```

```
##                     mpg cyl  disp  hp drat    wt  qsec
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98
```

Some `tidyverse` functions

- Use `dplyr` for data manipulation functions
  - Each "verb" performs an action on a data set
  - Functionality similar to SQL
- `select` - select/reorder/rename columns (by name)
- `mutate` - create a column
- `filter` - subset by logical conditions
- `arrange` - sorting
- `group_by` - group by a factor/character
- Many others!

A data manipulation exampe

```
mtcars %>%
  rownames_to_column() %>%
  filter(cyl == 4) %>%
  group_by(gear) %>%
  filter(mpg == min(mpg)) %>%
  select(rowname, gear, mpg, cyl) %>%
  ungroup
```

```
## # A tibble: 3 × 4
##          rowname  gear   mpg   cyl
##            <chr> <dbl> <dbl> <dbl>
## 1 Toyota Corona     3  21.5     4
## 2 Porsche 914-2     5  26.0     4
## 3    Volvo 142E     4  21.4     4
```

purrr

- purrr - functional programming toolkit for R
- A good tutorial: http://ijlyttle.github.io/isugg_purrr/presentation.html
- Goal: we want to fit a linear regression for each cylinder type in mtcars
- Use nest (from tidyr) and map

```
unique(mtcars$cyl)
```

```
## [1] 6 4 8
```

purrr

Creating a nested data frame

```
mt_nest <- mtcars %>%
  nest(-cyl)
mt_nest
```

```
## # A tibble: 3 × 2
##     cyl                data
##   <dbl>              <list>
## 1     6  <tibble [7 × 10]>
## 2     4 <tibble [11 × 10]>
## 3     8 <tibble [14 × 10]>
```

purr

Using map to fit a regression for each cylinder type

```
mt_fit <- mt_nest %>%
  mutate(lm_fit = map(data, ~(lm(mpg ~ wt, data=.))))
mt_fit
```

```
## # A tibble: 3 × 3
##     cyl                data   lm_fit
##   <dbl>              <list>   <list>
## 1     6  <tibble [7 × 10]> <S3: lm>
## 2     4 <tibble [11 × 10]> <S3: lm>
## 3     8 <tibble [14 × 10]> <S3: lm>
```

purr

Using `map_dbl` to append the AIC of each model

```
mt_fit2 <- mt_fit %>%
  mutate(AIC = map_dbl(lm_fit, AIC))
mt_fit2
```

```
## # A tibble: 3 × 4
##     cyl                 data   lm_fit      AIC
##   <dbl>               <list>   <list>    <dbl>
## 1     6  <tibble [7 × 10]> <S3: lm> 25.65036
## 2     4 <tibble [11 × 10]> <S3: lm> 61.48974
## 3     8 <tibble [14 × 10]> <S3: lm> 63.31555
```

purrr

Plot the fitted values vs. residuals of each model

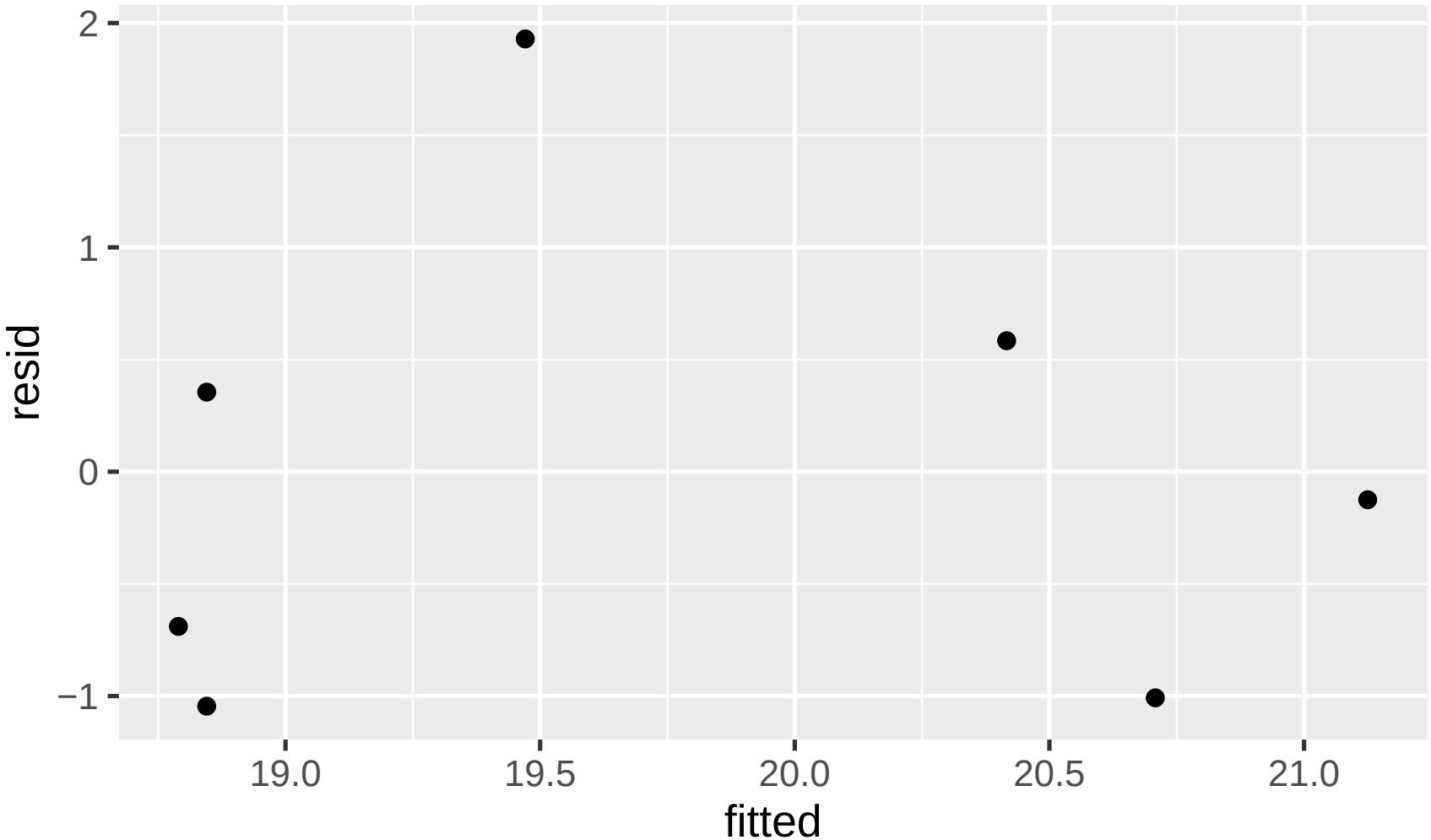
```
mt_plot <- mt_fit2 %>%
  mutate(resid_df =
           map(lm_fit,
               ~(data_frame(fitted=.$fitted.values,
                            resid=.$residuals))),
         pt = map(resid_df, ~(ggplot(.) +
                                geom_point(aes(x=fitted,
                                               y=resid)))))
mt_plot %>% select(-data, -lm_fit, -AIC)
```

```
## # A tibble: 3 × 3
##     cyl           resid_df       pt
##   <dbl>             <list>   <list>
## 1     6  <tibble [7 × 2]> <S3: gg>
## 2     4 <tibble [11 × 2]> <S3: gg>
## 3     8 <tibble [14 × 2]> <S3: gg>
```

purrr

Plot the fitted values vs. residuals of the first model (`cyl = 6`)

```
mt_plot$pt[[1]]
```



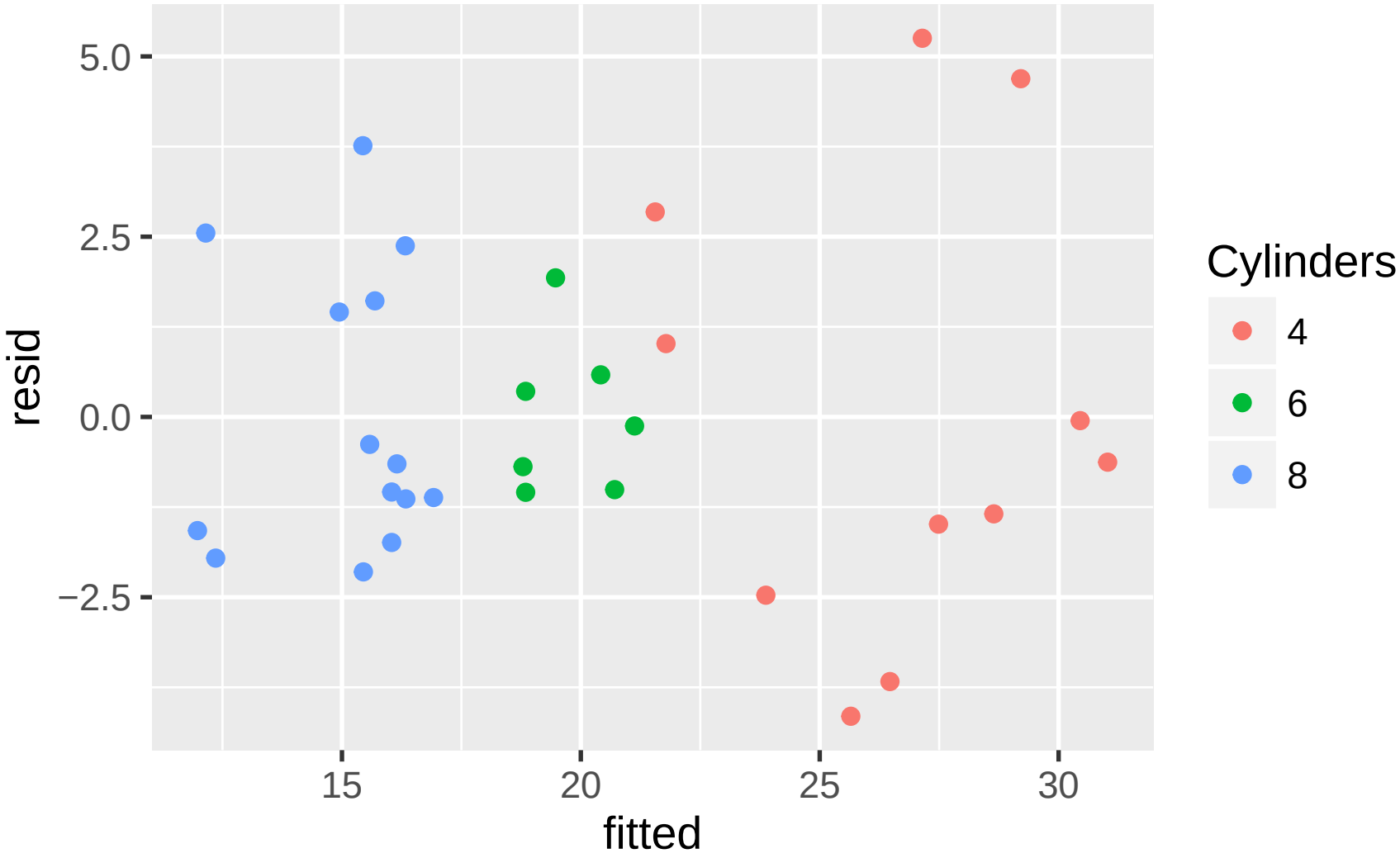
purrr

Or. . . get all of the fitted values/residuals on one plot

```
mt_plot2 <- mt_fit %>%
  mutate(fit_resid =
           map(lm_fit,
               ~data_frame(fitted=.$fitted.values,
                                          resid=.$resid)))
  unnest(data, fit_resid) %>%
  ggplot() +
  geom_point(aes(x=fitted,
                 y=resid,
                 color=as.factor(cyl))) +
  labs(color = 'Cylinders')
```

purrr

```
mt_plot2
```



Shiny

- A web application framework for R: `https://shiny.rstudio.com`
- Analyses and plots can be turned into *interactive* applications
- Some new syntax, but all analysis/plotting "machinery" is standard R code

Some example Shiny apps

- Free app hosting services: `https://www.shinyapps.io`
- k-means: `https://shiny.rstudio.com/gallery/kmeans-example.html`
- Outlier detection: `https://htran.shinyapps.io/daily_plot/`
- GAMM (by Becca Krouse): `https://becca-krouse.shinyapps.io/GAMMapp/`

plotly for interactive plots

- Sometimes we want to embed an interactive plot directly into an `rmarkdown` document
- Use plotly: `https://plot.ly/ggplot2/`
- Less labor intensive but less customization than Shiny
- Show plotly version of volcano plot in `rmarkdown` document (includes demonstration of `tidyverse` functions)