R Formula Interface

and Design Matrices

SYS 6018 | Spring 2021

Rfmla.pdf

```
#-- Required Packags
library(splines)
library(tidyverse)
```

1 Raw input data

The raw input data is often in the form of a data frame (or tibble). For example,

```
#-- Raw Input Data
#  cat is categorical with 3 levels: A,B,C
#  num is numerical
#  y is numerical response variable

Z = tibble(cat=c('A','A','B','B','C','C'), num=1:6, y=rnorm(6))
Z
#> # A tibble: 6 x 3
#>   cat     num      y
#>   <chr> <int>  <dbl>
#> 1 A         1 -0.493
#> 2 A         2  0.0860
#> 3 B         3  0.166
#> 4 B         4  0.519
#> 5 C         5 -1.17
#> 6 C         6  1.92
```

has three columns, `cat` is categorical data, `num` which is numerical data, and `y` which is the response variable.

2 Formula in models

The formula interface in R allows you to make transformations of the input data frame automatically. For example, categorical (or factor) columns will generate the appropriate dummy variables.

```
lm(y~cat, data=Z)$coef
#> (Intercept)        catB        catC
#>     -0.2036      0.5464      0.5776
lm(y~cat - 1, data=Z)$coef   # remove intercept
#>    catA    catB    catC
#> -0.2036  0.3428  0.3740
```

The default behavior is to convert categorical data to a *factor* and drop the first level.

The formula interface is easy to use:

```
#- numerical data only
lm(y~num, data=Z)$coef
#> (Intercept)         num
#>     -0.6939      0.2471
```

```
#- transformations
lm(y~log(num), data=Z)$coef
#> (Intercept)    log(num)
#>     -0.5236      0.6336
```

```
#- use I() to make custom functions
lm(y~I(3*num), data=Z)$coef
#> (Intercept)  I(3 * num)
#>    -0.69385     0.08237

#- we have already seen poly()
lm(y~poly(num, degree = 3), data=Z)$coef
#>             (Intercept) poly(num, degree = 3)1 poly(num, degree = 3)2
#>                 0.1711                 1.0338                 0.5994
#> poly(num, degree = 3)3
#>                 1.4527

#- how about B-splines
library(splines)
lm(y~bs(num), data=Z)$coef
#> (Intercept)    bs(num)1    bs(num)2    bs(num)3
#>     -0.6611      3.7151     -3.0315      2.3184

#- two predictors
lm(y~cat + num, data=Z)$coef
#> (Intercept)        catB        catC         num
#>      -2.218      -2.139      -4.794       1.343
lm(y~cat + num - 1, data=Z)$coef
#>   catA   catB   catC    num
#> -2.218 -4.357 -7.012  1.343

#- a:b stands for interactions
lm(y~cat + num + cat:num, data=Z)$coef
#> (Intercept)        catB        catC         num    catB:num    catC:num
#>     -1.0721      0.1804    -15.5873      0.5791     -0.2263      2.5179

#- use . to represent everything in data
lm(y~., data=Z)$coef
#> (Intercept)        catB        catC         num
#>      -2.218      -2.139      -4.794       1.343
lm(y~. - num, data=Z)$coef   # use . to include all, then remove some
#> (Intercept)        catB        catC
#>     -0.2036      0.5464      0.5776
```

2.1 model.matrix()

Behind the scenes, `lm()` is calling the function `model.matrix()` to construct the *design matrix*. The design matrix is the real valued X matrix used for calculating the coefficients. You have to pass a `formula` object into `model.matrix()`.

```
fmla = formula(y~num+cat)
model.matrix(fmla, data=Z)
#>   (Intercept) num catB catC
#> 1           1   1    0    0
#> 2           1   2    0    0
#> 3           1   3    1    0
#> 4           1   4    1    0
#> 5           1   5    0    1
#> 6           1   6    0    1
#> attr(,"assign")
#> [1] 0 1 2 2
#> attr(,"contrasts")
#> attr(,"contrasts")$cat
#> [1] "contr.treatment"
```

```
fmla = formula(y~num+cat-1)  # remove intercept
model.matrix(fmla, data=Z)
#>   num catA catB catC
#> 1   1    1    0    0
#> 2   2    1    0    0
#> 3   3    0    1    0
#> 4   4    0    1    0
#> 5   5    0    0    1
#> 6   6    0    0    1
#> attr(,"assign")
#> [1] 1 2 2 2
#> attr(,"contrasts")
#> attr(,"contrasts")$cat
#> [1] "contr.treatment"
```

Or, if you are good with data manipulation construct the design matrix manually.

```
library(dplyr)
Z %>%
  transmute(intercept=1,
           x1=num, x2=num^2,
           x3=ifelse(cat=='B',1,0), x4=ifelse(cat=='C',1,0)) %>%
  as.matrix()
#>      intercept x1 x2 x3 x4
#> [1,]         1  1  1  0  0
#> [2,]         1  2  4  0  0
#> [3,]         1  3  9  1  0
#> [4,]         1  4 16  1  0
#> [5,]         1  5 25  0  1
#> [6,]         1  6 36  0  1
```

Some functions (e.g., `glmnet`) do not take formulas so you will have to pass in the design matrix X directly. Another word of caution, some functions (again like `glmnet`) add the intercept automatically so you should not include a columns of ones.

The function `lm.fit()` fits a linear model from a design matrix:

```
X = model.matrix(formula(y~num+cat), data=Z)
Y = Z$y
lm.fit(x=X, y=Y)$coef
#> (Intercept)         num        catB        catC
#>      -2.218       1.343      -2.139      -4.794
```

2.2 Comparison

It is always good to compare the approaches just to make sure there are no mistakes.

```r
fmla = formula(y~num+cat + I(num^2) + sqrt(num))

#- lm()
beta.lm = lm(fmla, data=Z)$coef

#- lm.fit()
X = model.matrix(fmla, data=Z)
beta.lmfit = lm.fit(X, Z$y)$coef

#- direct matrix operations
beta.eq = solve(t(X) %*% X) %*% t(X) %*% Z$y

#- output
tibble(beta.lm, beta.lmfit, beta.eq) %>% knitr::kable()
```

beta.lm	beta.lmfit	beta.eq
-17.7859	-17.7859	-17.7859
-16.2163	-16.2163	-16.2163
0.3673	0.3673	0.3673
-2.9017	-2.9017	-2.9017
1.1275	1.1275	1.1275
32.3816	32.3816	32.3816