Stat 405/705
Class 4
Statistical computing with R

Richard P. Waterman

Wharton

Table of contents I

Today's module

Topics to be covered in this module:

- Last time
- Joining data frames
- The core modeling functions, `lm` and `glm`
- Functions used in today's class
- Next time

Last time

- Data frames
- Reading data from various sources

Joining data frames

- It is very common to need to combine data sources into a single analysis dataset
- Reasons:
  1. They come from different data bases (e.g. customer v. financials)
  2. They are created at different points in time
- The key to successful joining is that there is a **key** that is common to both data sets
- Examples: social security number, DEA number etc.
- If there isn't a unique key, then there will need to be a *fuzzy* match which is a potential time-sink/nightmare.

Joining data frames: example

- We have two datasets, one containing the predictor variables for an analysis and the other containing the y-variable
- They do have a unique identifier, called AccountID

```
# Get the two datasets from the web
x.data <- read.csv(file = "http://mathmba.com/richardw/x_var_join.csv")
y.data <- read.csv(file = "http://mathmba.com/richardw/y_var_join.csv")
```

Joining data frames: example

```
# Have a look at the two data sets
head(x.data,3)

##   SocialMediaIndex WordPress ACCOUNTID
## 1                1       YES     INF85
## 2                0        NO     JEM10
## 3               12        NO     CYN02

head(y.data,3)

##   AccountID Status
## 1     NDE65      1
## 2     INF85      1
## 3     JEM10      0
```

The goal is now to join these two data sets

Joining data frames: example

The command needed is `merge`. It takes a variety of arguments which provide control over how the joining is done. What should we do with cases that don't match? What should we do with duplicate matches? what should we do with duplicate column names and so on?

```
# A first attempt (it does not work)!
merge(x = x.data, y = y.data, by = "AccountID")

## Error in fix.by(by.x, x):  'by' must specify a uniquely valid column

#The merge argument is case sensitive and accountID
#doesn't exactly match across the two data frames.
```

Joining data frames: example

We could rename the account ID column so it matches exactly, or we can specify the matching columns explicitly:

```
# Carefully specifying the matching column names
my.merge <- merge(x = x.data, y=y.data,
        by.x = "ACCOUNTID", by.y = "AccountID")
print(my.merge)

##   ACCOUNTID SocialMediaIndex WordPress Status
## 1     ASC53                5        NO      1
## 2     INF85                1       YES      1
## 3     JEM10                0        NO      0
## 4     QUT35               12       YES      0
## 5     XCT80               34        NO      0
```

This data frame has only those IDs that were observed in both data frames.

Joining data frames: example

You can also specify matching columns by their numeric column index (but I don't think it is a good idea as it will not be robust to changes like the addition of a new column):

```
# Merging by column index
my.merge <- merge(x = x.data, y=y.data,
        by.x = 3, by.y = 1)
print(my.merge)

##   ACCOUNTID SocialMediaIndex WordPress Status
## 1     ASC53                5        NO      1
## 2     INF85                1       YES      1
## 3     JEM10                0        NO      0
## 4     QUT35               12       YES      0
## 5     XCT80               34        NO      0
```

Joining data frames: example

If we wanted to keep all the accounts in the y data frame then the `all.` argument is applied:

```
# Keeping all the rows in the y data frame
my.merge <- merge(x = x.data, y=y.data,
        by.x = "ACCOUNTID", by.y = "AccountID",
        all.y = TRUE)
print(my.merge)

##   ACCOUNTID SocialMediaIndex WordPress Status
## 1     ASC53                5        NO      1
## 2     INF85                1       YES      1
## 3     JEM10                0        NO      0
## 4     QUT35               12       YES      0
## 5     XCT80               34        NO      0
## 6     NDE65               NA      <NA>      1
## 7     XRO15               NA      <NA>      0
```

This data frame has all the IDs that were observed in the y data frame.

Joining data frames: example

If we wanted to keep all the accounts in both the x and y data frames then use both `all.x` and `all.y`:

```
# Keeping all the rows in the x and y data frames
my.merge <- merge(x = x.data, y=y.data,
        by.x = "ACCOUNTID", by.y = "AccountID",
        all.x = TRUE, all.y = TRUE)
print(my.merge)
```

```
##    ACCOUNTID SocialMediaIndex WordPress Status
## 1      ASC53                5        NO      1
## 2      CYN02               12        NO     NA
## 3      DIU03               76        NO     NA
## 4      INF85                1       YES      1
## 5      JEM10                0        NO      0
## 6      PYT29               21        NO     NA
## 7      QUT35               12       YES      0
## 8      WCV02               18        NO     NA
## 9      XCT80               34        NO      0
## 10     NDE65               NA      <NA>      1
## 11     XRO15               NA      <NA>      0
```

Regression in R

- The command used to run a regression in R is called `lm` for *linear model*.
- To make use of it you need to understand:
  1. The R syntax for specifying a model
  2. The additional functions available for using the results of a regression (residuals, predictions etc.)
- We will do a simple regression, a multiple regression, some plotting and prediction.
- Many of the commands we will use are called "generic" functions and can be used on other types of models too.

Regression in R

Obtain the data set "ProdTime.csv".

```
prodtime <- read.csv(file = "http://mathmba.com/richardw/ProdTime.csv")
summary(prodtime)

##   Time.for.Run   Manager    Run.Size
##  Min.   :147.0   a:20    Min.   : 58.0
##  1st Qu.:207.8   b:20    1st Qu.:143.8
##  Median :225.5   c:20    Median :208.0
##  Mean   :227.7           Mean   :209.3
##  3rd Qu.:252.0           3rd Qu.:281.0
##  Max.   :304.0           Max.   :345.0
```

The modeling syntax

- We will write a *formula* to specify a model
- The outcome variable (y) is on the left-hand side
- The outcome and predictor variables are separated by "~"
- The predictor (x) variables are added to the right-hand side

The modeling syntax

- A simple regression formula for Time.for.Run against Run.Size:

  `Time.for.Run ~ Run.Size`

- If you have multiple predictor terms, because it is a linear model which is additive, they are joined with the "+" sign.
- A multiple regression formula for Time.for.Run against Run.Size and Manager:

  `Time.for.Run ~ Run.Size + Manager`

- Interactions between variables are indicated with a colon ":".
- A multiple regression formula for Time.for.Run against Run.Size, Manager and the interaction between the two:

  `Time.for.Run ~ Run.Size + Manager + Run.Size:Manager`

Running the simple regression

The "data" argument references a data frame in which to find the variables.

```
lm.out <- lm(Time.for.Run ~ Run.Size, data = prodtime)
```

- Two important functions that can be applied to an `lm` model object (the output from a regression) are `summary` and `plot`.
- These are called `generic` functions because they can be applied to lots of different types of objects.
- To find out about them specifically for `lm` objects, enter `help(summary.lm)` and `help(plot.lm)`.

The *summary* command

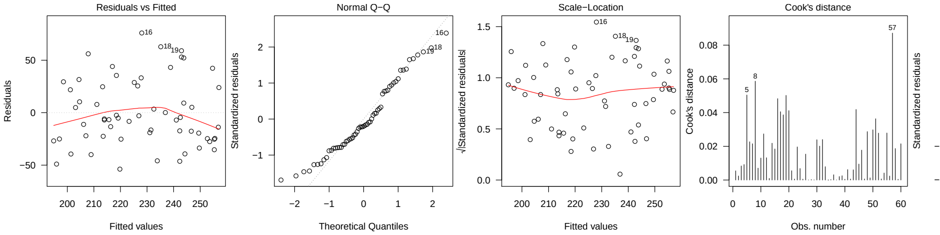
```r
summary(lm.out)
```

```
##
## Call:
## lm(formula = Time.for.Run ~ Run.Size, data = prodtime)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -53.784 -24.568  -6.302  24.911  75.985
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 182.31371   10.96144  16.632  < 2e-16 ***
## Run.Size      0.21659    0.04848   4.468 3.72e-05 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 32.11 on 58 degrees of freedom
## Multiple R-squared:  0.256, Adjusted R-squared:  0.2432
```

The *plot* command

```r
#There are multiple diagnostic plots and you
#can specify which ones you want with the "which" argument

plot(lm.out,which=c(1:5))
```



Extracting from the regression summary

```
#You can pull individual pieces from the regression summary
reg.summary <- summary(lm.out) #Save the whole summary
names(reg.summary) # What's in the summary?

##  [1] "call"          "terms"         "residuals"
##  [4] "coefficients"  "aliased"       "sigma"
##  [7] "df"            "r.squared"     "adj.r.squared"
## [10] "fstatistic"    "cov.unscaled"

reg.summary$coefficients # The coefficients table

##                Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept) 182.3137059 10.96143847 16.632279 9.662954e-24
## Run.Size      0.2165919  0.04847911  4.467737 3.717235e-05

reg.summary$coefficients[,4] # Just the p-values

##  (Intercept)     Run.Size
## 9.662954e-24 3.717235e-05
```

Using the subset argument to lm

The `subset` argument restricts the analysis to a subset of the data. The subset needs to be a vector (either logical or row indices) that indicates which rows of the data frame to use. We will rerun the regression excluding Manager c.

```
lm.out.no.c <- lm(Time.for.Run ~ Run.Size, data = prodtime,
        subset = (Manager != "c"))
```

Using the subset argument to lm

```r
summary(lm.out.no.c)
```

```
##
## Call:
## lm(formula = Time.for.Run ~ Run.Size, data = prodtime, subset = (Manager
##     "c"))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -56.463 -25.245  -7.601  20.997  63.888
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 206.10692   14.24401  14.470   <2e-16 ***
## Run.Size      0.16116    0.06274   2.569   0.0143 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 31.08 on 38 degrees of freedom
```

Including a categorical variable

This is a "parallel lines" regression in Stat613/621 parlance.

```
lm.out.manager <- lm(Time.for.Run ~ Run.Size + Manager, data = prodtime)
```

Including a categorical variable

```r
summary(lm.out.manager)
```

```
##
## Call:
## lm(formula = Time.for.Run ~ Run.Size + Manager, data = prodtime)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.979 -12.467   0.765  12.041  37.531
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 215.11848    6.14820  34.989  < 2e-16 ***
## Run.Size      0.24337    0.02508   9.705 1.34e-13 ***
## Managerb    -53.06082    5.24159 -10.123 2.93e-14 ***
## Managerc    -62.16817    5.18003 -12.002  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Including a categorical variable

- The default coding scheme for the categorical variable is to treat the first level (manager a) as the baseline and report differences from this baseline level.
- Manager b sets up their machine 53 minutes faster than manager a, etc.

By way of comparison, JMP uses the last level of the categorical variable as a baseline.

Indicator Function Parameterization

Term	Estimate	Std Error	DFDen	t Ratio	Prob>|t|
Intercept	152.95031	6.245301	56.00	24.49	<.0001*
Run Size	0.243369	0.025076	56.00	9.71	<.0001*
Manager[a]	62.168171	5.180029	56.00	12.00	<.0001*
Manager[b]	9.1073536	5.224343	56.00	1.74	0.0868

Including a categorical variable

How long does it take `Manager c` to set up their machine? It has to be the same answer in R and JMP.

- In R, as Manager a is the baseline we do 215.118 - 62.16817 = 152.95.
- In JMP, manager **c** is the baseline so we just read off the intercept from the JMP output, 152.95.
- The answer is the same.

The partial-F tests

If you want to test a multi-level categorical, then `anova` applied to the model output object will do it:

```
anova(lm.out.manager)

## Analysis of Variance Table
##
## Response: Time.for.Run
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Run.Size   1  20577 20577.5  76.729 4.424e-12 ***
## Manager    2  44774 22387.0  83.477 < 2.2e-16 ***
## Residuals 56  15018   268.2
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the `manager` variable has 2 degrees of freedom and is highly significant (the p-value is under the column `Pr(>F)`).

Prediction in regression

If you just want to predict for the observed values of x, you can use `predict` on the output regression object:

```
predict(lm.out.manager) #The predicted values -- y^hats
```

```
##        1        2        3        4        5        6
## 264.7658 240.1855 249.9202 266.2260 296.4037 239.9421
##        7        8        9       10       11       12
## 278.6378 243.8360 263.3055 236.2916 233.3712 255.7611
##       13       14       15       16       17       18
## 247.4866 254.0575 299.0808 266.4693 283.2618 274.5005
##       19       20       21       22       23       24
## 283.2618 284.2353 234.3382 185.4211 219.9795 197.1028
##       25       26       27       28       29       30
## 199.5365 241.1526 212.1917 203.4304 218.5193 231.6612
##       31       32       33       34       35       36
## 242.1261 237.7454 237.5020 223.3866 229.7142 216.8157
##       37       38       39       40       41       42
## 202.7003 200.2666 188.8283 234.5816 195.5399 198.9470
##       43       44       45       46       47       48
## 236.6692 174.3668 180.6944 235.2090 218.9033 195.0531
```

Prediction in regression

It is usually much more interesting to predict on new values of x, for which we don't know y yet. To do this, we will apply `predict` to the lm output object and give it the "`newdata`" argument.
Put the new data into a data frame and offer it to the `predict` function:

```
# Predict for a manager "b" job at 132 items
# and a manager "c" job at 190 items
x.predict <- data.frame(Manager = c("b","c"), Run.Size = c(132, 190))
print(x.predict)

##   Manager Run.Size
## 1       b      132
## 2       c      190
```

Prediction in regression

```r
# Ask for the predictions using the "newdata" argument
my.predictions <- predict(lm.out.manager,newdata=x.predict)
print(my.predictions)

##        1        2
## 194.1824 199.1904

#We can get 95% prediction intervals with the "interval" argument
predict(lm.out.manager,newdata=x.predict,interval="prediction")

##        fit      lwr      upr
## 1 194.1824 160.2135 228.1512
## 2 199.1904 165.5695 232.8113
```

To find out more about predict use `help(predict.lm)` because `predict` is a generic function.

Logistic regression

- When the outcome variable (y-variable) is a two level categorical variable, and not a continuous one, the appropriate modeling technique is *logistic regression*.
- Logistic regression models the probability of a *success* in a dichotomous outcome.

Logistic regression

We will model the probability of a *voluntary job turnover* as a function of age and gender.

```
hr.data <- read.csv(file = "http://mathmba.com/richardw/HR.csv")
summary(hr.data)

##    EmployeeId        age           volturn
##  Min.   :   1   Min.   :20.00   Min.   :0.000
##  1st Qu.:1412   1st Qu.:32.00   1st Qu.:0.000
##  Median :2824   Median :38.00   Median :0.000
##  Mean   :2824   Mean   :38.98   Mean   :0.111
##  3rd Qu.:4236   3rd Qu.:45.00   3rd Qu.:0.000
##  Max.   :5647   Max.   :70.00   Max.   :1.000
##
##     Gender         npjc            salary
##  Female:2515   Min.   :0.0000   Min.   : 18162
##  Male  :3132   1st Qu.:0.0000   1st Qu.: 42650
##                Median :0.0000   Median : 54249
##                Mean   :0.0577   Mean   : 58047
##                3rd Qu.:0.0000   3rd Qu.: 71604
##                Max.   :1.0000   Max.   :145162
```

Logistic regression

- The y-variable is `volturn`. It is coded as $0/1$: $1 =$ Quit, $0 =$ Stay.
- The predictors are `age` and `Gender`.
- The command for *logistic regression* is `glm` which stands for Generalized Linear Model.

Logistic regression

- The name for a dichotomous random variable (the outcome here, quit or stay) is a *Bernoulli* random variable.
- A Bernoulli random variable is a special case of a *Binomial* random variable.
- We will need to tell the `glm` function we are dealing with a Binomial Random variable.
- A Bernoulli rv is like the outcome of the toss of a single coin: head = success, tail = failure.
- A Binomial rv is the number of successes in n independent coin tosses.

Logistic regression

```
# glm syntax is very similar to lm
glm.out <- glm(volturn ~ age + Gender, # The model specification
        data =hr.data, # The data frame in which to find the data
        family="binomial") # telling R that the outcome
                           # is a binomial random variable
```

Logistic regression

```
summary(glm.out) # Summarize the output

##
## Call:
## glm(formula = volturn ~ age + Gender, family = "binomial", data = hr.dat
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.6667  -0.5234  -0.4687  -0.4043   2.5425
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.687294   0.201631  -3.409 0.000653 ***
## age         -0.033507   0.005194  -6.452 1.11e-10 ***
## GenderMale  -0.226261   0.085115  -2.658 0.007854 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

Logistic regression

What did we learn from the logistic regression model?

- Looking at the signs of the coefficients, older people are less likely to quit. Males are less likely to quit than females.
- Both effects are significant.
- `glm` has its own predict method, just like *lm*. Use `help(predict.glm)` to find out more about it.

Module summary

Topics covered today include:

- Joining data frames
- The `lm` and `glm` commands

Next time

- Writing your own functions (I)

Today's function list

Do you know what each of these functions does?

```
anova
glm
lm
merge
pie
plot
predict
summary
```