STATS 401. Applied Statistical Methods II

1. Introduction

Welcome!

Objectives: Linear statistical models are the foundation for most of applied statistics. We will develop statistical computation skills (R programming) and mathematical skills (working with matrices) while studying data analysis using linear models.

Pre-requisites: We will assume familiarity with material in STATS 250. All course notes and labs for STATS 250 are at

`open.umich.edu/find/open-educational-resources/statistics`

If you have a different background (AP Statistics, STATS 280, or some other introductory statistics class) you should check the STATS 250 notes and if necessary come for help in office hours.

Let's get started

We will work through a data analysis using a linear model, and then study the math and stats so that (i) we can command the computer to generate what we want; (ii) we can interpret what the computer tells us.

- Obtain the data, usually from the internet
- Install R (`www.r-project.org`) and Rstudio (`www.rstudio.com`)
- Read the data into R
- Plot the data
- Develop a model
- Estimate parameters and test hypotheses of interest
- Interpret the results

The two rising stars in statistical computing are R and Python (`http://r4stats.com/articles/popularity/`). Generally, R is preferred for data analysis, and Python for larger programming projects.

We live in an era of abundant data. Learn R!

Case study: Are people healthier in booms or busts?

- Is population health **pro-cyclical** (improving in business cycle booms) or **counter-cyclical** (improving in recessions), or neither?
- **Life expectancy at birth** combines instantaneous death rates at all ages and is a basic measure of current population health.
- USA data for 1933–2015 are in the file `life_expectancy.txt` on the course GitHub repository `github.com/ionides/401w18/01` or the website `ionides.github.io/401w18/01`. The first lines of this file are:

```
# The United States of America, Life expectancy at birth.
# Downloaded from Human Mortality Database on 30 Oct 2017.
# HMD request that you register at http://www.mortality.org
# if you use these data for research purposes.
  Year        Female    Male     Total
  1933         62.78    59.17    60.88
  1934         62.34    58.34    60.23
```

- Note: # denotes a comment in R, so the first four text lines will be ignored when we read in the data.

Read the data into R and then inspect it

```
L <- read.table(file="life_expectancy.txt",header=TRUE)
```

Question 1.1. Why should we prefer to use the command line form of R rather than a menu option, say in R Commander?

Now, let's check on the data. To see the first three rows,

```
L[1:3,]
```

```
##   Year Female  Male Total
## 1 1933  62.78 59.17 60.88
## 2 1934  62.34 58.34 60.23
## 3 1935  63.04 58.96 60.89
```

Here, we're using **matrix indexing**. `L[i,j]` is the row `i` column `j` entry of `L`. Also, `1:3` is the sequence `1,2,3` and the blank space after the comma in `L[1:3,]` requests all the rows for the specified columns.

Matrices and their dimensions

Mathematically, we write $\mathbb{L} = \begin{bmatrix} \ell_{11} & \ell_{12} & \dots & \ell_{1n} \\ \ell_{21} & \ell_{22} & \dots & \ell_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{m1} & \ell_{m2} & \dots & \ell_{mn} \end{bmatrix}$.

We say $\mathbb{L}$ is a matrix with **dimension** $m \times n$. To get the dimension in R,

```
dim(L)
```

```
## [1] 83  4
```

We can also get the number of rows and columns separately,

```
cat("number of rows = ", nrow(L),
  "; number of columns = ", ncol(L))
```

```
## number of rows =  83 ; number of columns =  4
```

Extracting rows and columns from a matrix

A single row or column of a matrix is a **vector**. Vectors will be discussed more in Chapter 2.

For example, we can set `y` to be total life expectancy, combining men and women, which is the fourth column of `L`, as follows.

```
y <- L[,4]
y[1:3]
```

```
## [1] 60.88 60.23 60.89
```

Question 1.2. We read the assignment operator `<-` as "`y` gets `L[,4]`". We could have written `y=L[,4]`. However, `<-` is slightly better coding practice than =. Why?

Vectors in R

For R, vectors are not matrices. A vector has a `length` but not a `dim`. When subsetting a matrix, the dimension of length `1` is **dropped**.

```
dim(y)

## NULL
```

```
length(y)

## [1] 83
```

We can extract the components of a vector. For example, to obtain the increase in life expectancy each year over the previous year,

```
g <- y[2:length(y)] - y[1:(length(y)-1)]
```

Since the increase is not defined for the first year life expectancy is measured, let's set the first increase to `NA`,

```
g <- c(NA,g)
g[1:8]

## [1]    NA -0.65  0.66 -0.54  0.70  1.34  0.68  0.16
```

Note: here we've seen two of R's special non-numeric values. `NULL` means "doesn't exist". `NA` means "not available" or "missing". Data matrices can have `NA` entries but not `NULL`. R tries to treat missing data appropriately.

Numeric, logical and character data in R

Numeric data are matrices and vectors whose entries are numbers. **Qualitative data** are **character strings**. **Logical data** are `TRUE` or `FALSE`.

```
g[1:4]

## [1]    NA -0.65  0.66 -0.54
```

```
L_up_logical <- g>0
L_up_logical[1:4]

## [1]    NA FALSE  TRUE FALSE
```

```
L_up_qualitative <- ifelse(g>0,"increased","decreased")
L_up_qualitative[1:4]


## [1] NA          "decreased" "increased" "decreased"
```

The `class` function tells us what data type R is working with

```
class(g)

## [1] "numeric"
```

```
class(L_up_logical)

## [1] "logical"
```

```
class(L_up_qualitative)

## [1] "character"
```

Getting help with R

Learning a computing language is sometimes frustrating. Please proceed in the following order

1. The R help, e.g., type `?ifelse` for information on the syntax of `ifelse`.
2. The internet, e.g., google "R ifelse".
3. Classmates.
4. Office hours, start-and-end of class, lab
5. Email to instructor and/or GSI.

For detailed email help, please construct and email a simple example demonstrating the issue. Sometimes, the issue gets resolved by writing it out!

R data structures: dataframes and matrices

- A matrix in R must have all entries of the same type. The mathematics of fitting a linear statistical model will require type to be numeric.
- For example, to convert data to a numeric representation for statistical analysis, `L_up_logical` or `L_up_qualitative` could be coded using `0` for `FALSE` (or `"decreased"`) and `1` for `TRUE` (or `"increased"`).
- A dataframe in R may have different types in each column. Data are usually stored in dataframes, e.g., `read.table()` generates a dataframe.

```
class(L)

## [1] "data.frame"
```

```
L_matrix <- as.matrix(L)
class(L_matrix)

## [1] "matrix"
```

- For many purposes, dataframes and matrices behave the same.

Innuit have many words for snow (`wikipedia:Eskimo_words_for_snow`) and R has many ways of working with data. To do effective data analysis, these are worth learning!

Subsetting matrices and vectors in R

- Vectors and matrices can be subsetted using logical vectors. Each entry of a vector (or row/column of a matrix) is included if the logical vector is `TRUE` and excluded if `FALSE`.
- Rows and columns can be selected using row and column names:

```
colnames(L)

## [1] "Year"   "Female"
## [3] "Male"   "Total"
```

```
rownames(L)[1:8]

## [1] "1" "2" "3" "4"
## [5] "5" "6" "7" "8"
```

Question 1.3. What is computed below. Can you find any interpretation?

```
L[g<0,"Year"]


##  [1]   NA 1934 1936 1943 1957 1960 1962 1963 1966 1968 1980
## [12] 1985 1988 1993 2015
```

Building matrices and vectors in R

The `c()` function combines numbers into vectors, and also combines vectors into longer vectors.

```
u <- c(1,2)
u

## [1] 1 2
```

```
v <- c(3,4)
v

## [1] 3 4
```

```
w <- c(u,v)
w

## [1] 1 2 3 4
```

We can build a matrix using `matrix()`. Also, we can get a matrix by binding together vectors either as rows or columns.

```
A <- matrix(1:6,nrow=2)
A

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
B <- rbind(u,v)
B

##   [,1] [,2]
## u    1    2
## v    3    4
```

```
C <- cbind(u,v)
C

##      u v
## [1,] 1 3
## [2,] 2 4
```

Exercises. What would `cbind(A,B)` produce? Play with these functions. Check out `?matrix` to get the syntax of this command.

Continuing our health economics case study

We looked at data on mortality. We'll use Bureau of Labor Statistics data on unemployment as a measure of the business cycle.

```
# Data extracted on: February 4, 2016
# from http://data.bls.gov/timeseries/LNU04000000
# Percent unemployment, age 16+, not seasonally adjusted
Year,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
1948,4.0,4.7,4.5,4.0,3.4,3.9,3.9,3.6,3.4,2.9,3.3,3.6
1949,5.0,5.8,5.6,5.4,5.7,6.4,7.0,6.3,5.9,6.1,5.7,6.0
```

```
U <- read.table(file="unemployment.csv",sep=",",header=TRUE)
U[1:2,]


##   Year Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1 1948   4 4.7 4.5 4.0 3.4 3.9 3.9 3.6 3.4 2.9 3.3 3.6
## 2 1949   5 5.8 5.6 5.4 5.7 6.4 7.0 6.3 5.9 6.1 5.7 6.0
```

Note: the data are in a comma separated variable (csv) format, so we use `read.table(...,sep=",",...)`.

Averaging columns in R

We want annual average unemployment. For each row, we must average columns `2:13`.

```
u <- apply(U[,2:13],1,mean)
u[1:6]
```

```
## [1] 3.766667 5.908333 5.325000 3.333333 3.033333 2.925000
```

- `apply()` carries out an operation (here, taking the mean) on rows or columns of matrices. We will learn more about it later.
- The middle argument 1 to `apply()` asks for the function `mean()` to be applied to each row.
- Setting 2 would give the average over rows for each column.
- Remember: `apply(U,1,...)` gives a vector of length `dim(U)[1]`, and `apply(U,2,...)` gives a vector of length `dim(U)[2]`.

```
dim(U)
```

```
## [1] 68 13
```

```
length(apply(U,1,mean))
```

```
## [1] 68
```

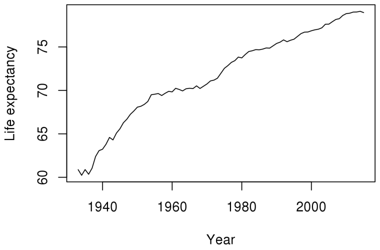
```
length(apply(U,2,mean))
```

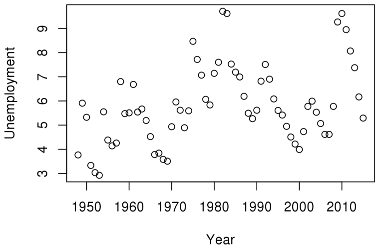
```
## [1] 13
```

Plotting the data

```
plot(L$Year,y,type="line",
  xlab="Year",
  ylab="Life expectancy")
```

```
plot(U$Year,u,
  xlab="Year",
  ylab="Unemployment")
```





- A basic rule of applied statistics is to plot the data.
- Carefully designed plots can reveal secrets in the data: (i) label axes; (ii) lines or points or both; (iii) any other creative ideas?
- This course will use the basic `plot()` function. A powerful modern approach to graphics is the “grammar of graphics” in the `ggplot2` package, taught in STATS 306.

Detrending life expectancy

- Life expectancy is generally increasing with time. We say it has an **increasing trend**.
- We're interested in whether it is above or below trend during economic booms.
- Subtracting an estimate of the trend from each data point is called **detrending**. A basic way to do this is to fit a linear trend that fits the data best, by finding the line mimizing the sum of squares of distances to the data.
- Most of you have seen this done before: `https://open.umich.edu/sites/default/files/downloads/interactive_lecture_notes_12-regression_analysis.pdf`
- In this course, we're going to study linear models and their statistical properties in much more detail.
- First, let's see how to compute this **least squares** fitted line using the `lm()` function in R.

Fitting a linear model using lm()

```
L_fit <- lm(Total~Year,data=L)
```

- Using `Total~Year` to model *Total depends on Year* in R is called a **formula**. Type `?lm` in R to see the function description.
- We could have said `lm(L$Total~L$Year)` or `lm(y=L$Total,x=L$Year)`.
- Writing `data=L` tells `lm()` to look for the linear model variables in the dataframe `L` and makes the model easier to read.

```
plot(Total~Year,L,type="l")
lines(L$Year,L_fit$fitted.values,
  lty="dotted")
```

- The **fitted values** in `L_fit$fitted.values` give the dotted line.
- We use formulas in `plot()` just like we did in `lm()`.

Exploring the output of `lm`

- We call `L_fit` a **fitted model object** since it is an R object that was created by fitting a model, in this case a linear model fitted using `lm`.
- First, let's check the class of the object

```
class(L_fit)

## [1] "lm"
```

- We see that `lm` is both the name of the function to fit a linear model and the class of the resulting fitted model object.
- Now, let's see what the fitted model object contains:

```
names(L_fit)

##  [1] "coefficients"  "residuals"     "effects"
##  [4] "rank"          "fitted.values" "assign"
##  [7] "qr"            "df.residual"   "xlevels"
## [10] "call"          "terms"         "model"
```

- `L_fit` is a list with all the things R thinks we might want to know about the fitted linear model. Components are accessed using `$`. We have already seen the fitted values accessed using `L_fit$fitted.values`. We will use other components later in the course.

Computer software notation vs math notation

- Computers compute things. That's what they do. It seems obvious.
- A computer function takes numbers in and spits numbers out. It can't know whether the analysis is correct, or reasonable, or useful for some purpose, or complete nonsense. Artificial intelligence is not (yet) good at applied statistics!
- **For describing statistical assumptions, understanding the behavior of statistical tests, and defining statistical models, mathematics is a more appropriate language than computer code.**
- We have to learn to write about statistics using two different languages: mathematics and computing. We have to learn when each is appropriate.
- If all is well, the math helps us understand the computing and vice versa.
- We have already seen one example: matrices and vectors are simultaneously (i) mathematical objects, with certain mathematical rules and definitions; (ii) R objects which follow a set of rules inspired by the mathematics.
- How do we mathematically write down the statistical linear model that we fitted using `lm()`?

A linear model – the sample version

- Suppose our data are $y_1, y_2, \ldots, y_n$ and on each individual i we have p explanatory variables $x_{i1}, x_{i2}, \ldots, x_{ip}$. A linear model is

$$\text{(LM1)} \qquad y_i = b_1 x_{i1} + b_2 x_{i2} + \cdots + b_p x_{ip} + e_i \quad \text{for } i = 1, 2, \ldots, n$$

- This is a model for a particular sample $y_1, \ldots, y_n$. A basic task of statistics is to generalize from a sample to a population. We'll do that later.
- The **residual error** terms $e_1, \ldots, e_n$ in equation (LM1) include everything about the data $y_1, y_2, \ldots, y_n$ that cannot be explained by the **linear combination** of the explanatory variables.
- Using **summation notation** we can write the linear model for this sample in a more compact way,

$$\text{(LM2)} \qquad y_i = \sum_{j=1}^{p} x_{ij} b_j + e_i \qquad \text{for } i = 1, 2, \ldots, n$$

- We'll review summation notation in due course.

Applying the linear model to detrend life expectancy

- When we did `L_fit<-lm(Total~Year,data=L)` earlier, we fitted the linear model (LM1) with y_i being the total life expectancy for the ith year in the dataset (recall that total life expectancy means combining males and females) and x_{i1} being the corresponding year.
- To fit a linear trend, we also want an **intercept**, which we can write by setting $x_{i2} = 1$ for each year i.
- In this special case, with $p = 2$ variables and $x_{i2} = 1$, the model (LM1) is called the **simple linear regression** model,

$$\text{(SLR1)} \qquad y_i = b_1 x_{i1} + b_2 + e_i \quad \text{for } i = 1, 2, \ldots, n$$

- Here, b_2 is the intercept for the **fitted line** $y_i = b_1 x_{i1} + b_2$ when we ignore the residual errors $e_1, \ldots, e_n$.
- In `L_fit<-lm(Total~Year,data=L)`, we gave R the task of finding the values of the **coefficients** b_1 and b_2 which minimize the **sum of squared errors**, $\sum_{i=1}^{n} e_i^2$.
- We didn't have to tell R we wanted an intercept. By default, it assumed we did. In this case it was right.

Is unemployment assosciated with higher or lower mortality?

- Now, we'll fit another linear model to see if the detrended life expectancy can be explained by the level of economic activity, quantified by the unemployment rate.
- We have seen that `residuals` is one of the components of an `lm` object, by looking at `names(L_fit)`.
- **Residual** is a more polite name than **residual error**. That is appropriate here, since the "error" e_i is exactly the deviation from trend which we are most interested in. Interpretation of e_i depends on the exact situation under consideration.
- First, let's set up the variables for the regression. Since we detrended life expectancy, we should also detrend unemployment. Then, we have some work to do to make sure that the years for these two datasets match!

```
L_detrended <- L_fit$residuals
U_detrended <- lm(u~U$Year)$residuals
L_detrended <- subset(L_detrended,L$Year %in% U$Year)
```

A linear model linking mortality and unemployment

```
lm1 <- lm(L_detrended~U_detrended)
coef(lm1)


## (Intercept) U_detrended
##   0.2899928   0.1313673
```

- We have obtained a positive coefficient for the sample linear model. Higher unemployment seems to be associated with higher life expectancy. This may be surprising. Is the result **statistically significant**? What happens if we use a different explanatory variable instead of unemployment? Or if we use more than one explanatory variable? Are there any violations of statistical assumptions that might invalidate this analysis? Is it reasonable to make a causal interpretation (that economic cycles cause fluctuations in life expectancy) or must we limit ourselves to a claim that these variables are associated?
- Giving informed answers to statistical questions such as these is a primary goal of the course.