前言

开篇先来谈一谈，我所理解的编程之道。具体来说，将讨论怎么学习编程语言、R 语言简介、R 语言编程思想、本书的特色及内容安排。

温馨提示

本章为了阐述需要，会涉及到一些 R 语言代码，读者可以先忽略代码细节，只当它们是计算过程“翻译”而成，把关注点放在我想传达的编程思维上。

0.1 怎么学习编程语言？

编程语言是人与计算机沟通的一种形式语言，根据设计好的编程元素和语法规则，来严格规范地表达我们想要做的事情的每一步（程序代码），使得计算机能够明白并正确执行，得到期望的结果。

编程语言和数学语言很像，数学语言是最适合表达科学理论的形式语言，用数学符号、数学定义和逻辑推理，来规范严格地表达科学理论。

很多人说：“学数学，就是靠大量刷题”，“学编程，就是照着别人的代码敲代码”。

我不认可这种观点，这样的学习方法事倍功半，关键是这样做你学不会真正的数学，也学不会真正的编程！

那么应该怎么学习编程语言呢？

就好比要成为一个好的厨师，首先得熟悉各种常见食材的特点秉性，掌握各种基本的烹饪方法，然后就能根据客人需要随意组合食材和烹饪方法制作出各种可口的大餐。

数学的食材就是**定义**，烹饪方法就是**逻辑推理**，一旦你真正掌握了**定义 + 逻辑推理**，各种基本的数学题都不再话下，而且你还学会了数学知识。

同理，编程的食材和烹饪方法就是**编程元素**和**语法规则**，比如数据结构（容器）、分支/循环结构、自定义函数等，一旦你掌握了这些**编程元素**和**语法规则**，根据问题的需要，你就能信手拈来优化组合它们，从而自己写出代码解决问题。

所以，学习任何一门编程语言，根据我的经验，有这么几点建议（步骤）：

1. 理解该编程语言的核心思想，比如 Python 是面向对象，R 语言是面向函数也面向对象，另外，高级编程语言还都倡导向量化编程。在核心思想的引领下去学它去思考去写代码。

2. 学习该编程语言的基础知识，这些基础知识本质上是相通的同样的东西，只是在不同编程语言下批上了其特有的外衣（编程语法），基础知识包括：数据类型及数据结构（容器）、分支/循环结构、自定义函数、文件读写、可视化等。
3. 前两步完成之后，就算基本入门[1]了，可以根据需要，根据遇到的问题，借助网络搜索、借助帮助，遇到问题解决问题，逐步提升，用的越多会的越多，也越熟练。

以上是学习编程语言的正确、快速、有效的方法，切忌不学基础语法，用到哪就突击哪，找别人的代码一顿不知其所以然的瞎改，这样始终无法入门，更谈不上将来提高。

再来谈一个学编程过程中普遍存在的问题：如何跨越**能看懂别人的代码”到“自己写代码**的鸿沟。

绝大多数人在编程学习过程中，都要经历这样一个过程：

1. 学习基本语法
2. 能看懂和调试别人的代码

↓（编程之门）

3. 自己写代码

前两步没有任何难度，谁都可以做到。从第 2 步到第 3 步是一个“坎”，很多人困惑于此而无法真正进入编程之门。网上也有很多讲到如何跨越这步的文章和说法，但基本都是脱离实际操作的空谈，比如多敲书上的代码之类，治标不治本（只能提升编程基本知识）。

我的理念说白了也很简单，无非就是：**分解问题 + 实例梳理 + 翻译及调试**，具体来说，

- 将难以入手大问题分解为可以逐步解决的小问题
- 用计算机的思维去思考解决每步小问题
- 借助类比的简单实例和代码片段，梳理出详细算法步骤
- 将详细算法步骤用逐片段地用编程语法翻译成代码并调试通过

关于调试补充一点，可以说高级编程语言的程序代码就是逐片段调试出来的，借助简单实例按照算法步骤，从上一步的结果调试得到下一步的结果，依次向前推进直到到达最终的结果。还有一点经验之谈：**写代码时，随时跟踪关注每一步执行，变量、数据的值是否到达你所期望的值，非常有必要！**

这是我用数学建模的思维得出的科学的操作步骤。为什么大家普遍自己写代码解决具体问题时感觉无从下手呢？

这是因为你总想一步就从**问题**到**代码**，没有中间的过程，即使编程高手也做不到。

[1]至少要经历过一种编程语言的入门，再学习其它编程语言就会很快。

当然，编程高手可能能缩减这个过程，但不能省略这个过程。其实你平时看编程书是被欺骗了：编程书上只写问题（或者有简单分析）紧接着就是代码，给人的感觉就是应该从问题直接到代码，大家都是这么写出来的。其实不然。

所以，改变从**问题**直接到**代码**思维故式，按我上面说的步骤去操作，每一步是不是都不难解决。那么，自然就从无从下手，到能锻炼自己写代码了。

开始你可能只能自己写代码解决比较简单的问题，但是这种成就感再加上慢慢锻炼下来，你自己写代码的能力会越来越强，能解决问题也会越来越复杂，当然前提是，你已经真正掌握了基本编程语法，可以随意取用。当然二者也是相辅相成、共同促进和提高的关系。

好，说清了这个道理，下面拿一个具体的小案例来演示一下。

例 1.1 计算并绘制 ROC 曲线

ROC 曲线是二分类机器学习模型的性能评价指标，已知测试集或验证集每个样本真实类别及其模型预测概率值，就可以计算并绘制 ROC 曲线。

先来梳理一下问题，ROC 曲线是在不同分类阈值上对比真正率（TPR）与假正率（FPR）的曲线。

分类阈值就是根据预测概率判定预测类别的阈值，要让该阈值从 0 到 1 以足够小的步长变化，对于每个阈值 c，比如 0.85，则预测概率 ≥ 0.85 判定为 “Pos’’，< 0.85 判定为 “Neg”. 这样就得到了预测类别。

根据真实类别和预测类别，就能计算混淆矩阵：

		真实类别 y	
		$+$	$-$
预测类别 $\hat{y}$	$+$	TP（真正）	FP（假正）
	$-$	FN（假负）	TN（真负）

进一步就可以计算：

$$TPR = \frac{TP}{TP + FN}, \qquad FPR = \frac{FP}{FP + TN}$$

有一个阈值，就能计算一组 TPR 和 FPR，循环迭代都计算出来并保存。再以 FPR 为 x 轴，以 TPR 为 y 轴绘图，则得到 ROC 曲线。

于是，梳理一下经过分解后的问题：

- 让分类阈值以某步长在 $[1, 0]$ 上变化取值；
- 对某一个阈值，
  - 计算预测类别
  - 计算混淆矩阵

    - 计算 TPR 和 FPR
  - 循环迭代，计算所有阈值的 TPR 和 FPR
  - 根据 TPR 和 FPR 数据绘图

下面拿一个小数据算例，借助代码片段来推演上述过程。现在读者不用纠结于代码，更重要的是体会，自己写代码解决实际问题的这样一个思考过程。

```
library(tidyverse)


df = tibble(
  ID = 1:10,
  `真实类别`=c("Pos","Pos","Pos","Neg","Pos","Neg","Neg","Neg","Pos","Neg"),
  `预测概率`=c(0.95,0.86,0.69,0.65,0.59,0.52,0.39,0.28,0.15,0.06)
)
knitr::kable(df)
```

ID	真实类别	预测概率
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Neg	0.39
8	Neg	0.28
9	Pos	0.15
10	Neg	0.06

先来解决对某一个阈值，计算 TPR 和 FPR。以 $c = 0.85$ 为例。

计算预测类别，实际上就是 `If-else` 语句根据条件赋值，当然是用整洁的 `tidyverse` 来做。顺便多做一件事情：把类别变量转化为因子型，以保证 “Pos” 和 “Neg” 的正确顺序，与混淆矩阵中一致。

```
c = 0.85
df1 = df %>%
  mutate(
    `预测类别` = ifelse(预测概率 >= c, "Pos", "Neg"),
    `预测类别` = factor(预测类别, levels = c("Pos", "Neg")),
    `真实类别` = factor(真实类别, levels = c("Pos", "Neg"))
```

```
  )
knitr::kable(df1)
```

ID	真实类别	预测概率	预测类别
1	Pos	0.95	Pos
2	Pos	0.86	Pos
3	Pos	0.69	Neg
4	Neg	0.65	Neg
5	Pos	0.59	Neg
6	Neg	0.52	Neg
7	Neg	0.39	Neg
8	Neg	0.28	Neg
9	Pos	0.15	Neg
10	Neg	0.06	Neg

计算混淆矩阵，实际上就是统计交叉频数，本来为 “Pos” 预测为 “Pos” 的有多少，等等。用 R 自带的 `table()` 函数就能实现：

```
cm = table(df1$预测类别, df1$真实类别)
cm
```

```
##
##       Pos Neg
##   Pos   2   0
##   Neg   3   5
```

计算 TPR 和 FPR。根据其的计算公式，从混淆矩阵中取数计算即可。这里咱们再高级一点，用向量化计算来实现，毕竟高级编程语言提倡向量化嘛。关于向量化编程，说白了就是，对一列/矩阵/多维数组的数同时做同样的操作，既提升程序效率又大大简化代码。

向量化编程，关键是要用整体考量的思维来思考、来表示运算。比如这里计算 TPR 和 FPR，通过观察可以发现：混淆矩阵的第 1 行各元素，都除以其所在列和，正好是 TPR 和 FPR。

```
cm["Pos",] / colSums(cm)
```

```
## Pos Neg
## 0.4 0.0
```

这就完成了本问题的核心部分。接下来，要循环迭代对每个阈值，都计算一遍 TPR 和 FPR。用 `for` 循环当然可以，但咱们仍然更高级一点：泛函式编程。

先把上述计算封装为一个 **自定义函数**，该函数只要接受一个前文原始的数据框 `df` 和一个阈值 `c`，就能返回来你想要的 TPR 和 FPR。然后，再把该函数 **应用**到数据框 `df` 和一系列的阈值上，循环迭代自然就完成了。这就是 **泛函式编程**。

```
cal_ROC = function(df, c) {
  df = df %>%
  mutate(
    `预测类别` = ifelse(预测概率 >= c, "Pos", "Neg"),
    `预测类别` = factor(预测类别, levels = c("Pos", "Neg")),
    `真实类别` = factor(真实类别, levels = c("Pos", "Neg"))
  )
  cm = table(df$预测类别, df$真实类别)
  t = cm["Pos",] / colSums(cm)
  list(TPR = t[[1]], FPR = t[[2]])
}
```

测试一下这个自定义函数，能不能算出来刚才的梳理时结果：

```
cal_ROC(df, 0.85)
```

```
## $TPR
## [1] 0.4
##
## $FPR
## [1] 0
```

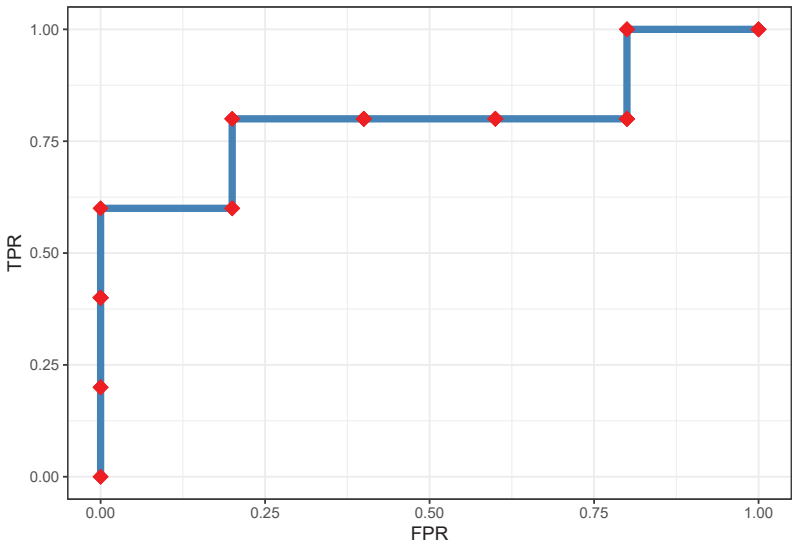
没问题，下面将该函数应用到一系列的阈值上（循环迭代），并一步到位将每次计算的两个结果按行合并到一起，这就彻底完成数据计算：

```
c = seq(1, 0, -0.02)
rocs = map_dfr(c, cal_ROC, df = df)
head(rocs)      # 查看前 6 个结果
```

```
## # A tibble: 6 x 2
##     TPR   FPR
##   <dbl> <dbl>
## 1   0       0
## 2   0       0
## 3   0       0
## 4   0.2     0
## 5   0.2     0
## 6   0.2     0
```

最后，用著名的 ggplot2 包绘制 ROC 曲线图形：

```
rocs %>%
  ggplot(aes(FPR, TPR)) +
  geom_line(size = 2, color = "steelblue") +
  geom_point(shape = "diamond", size = 4, color = "red") +
  theme_bw()
```



以上就是我所主张的学习编程的正确方法，我不认为照着别人的编程书敲代码是学习编程的好方法。

本节部分内容参阅Evaluation: Measures for Binary Classification: ROC visualization.

0.2 R 语言简介

0.2.1 什么是数据科学？

数据科学是综合了统计学、计算机科学和领域知识的交叉学科，其基本内容就是用数据的方法研究科学，用科学的方法研究数据（鄂维南院士）。

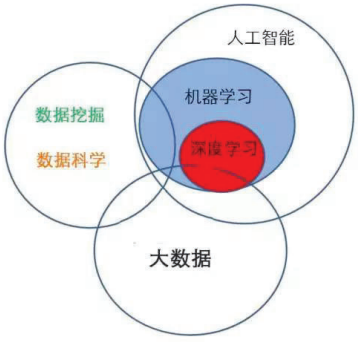


图 1: 数据科学的位置

Hadley Wickham 定义了数据科学的工作流程：

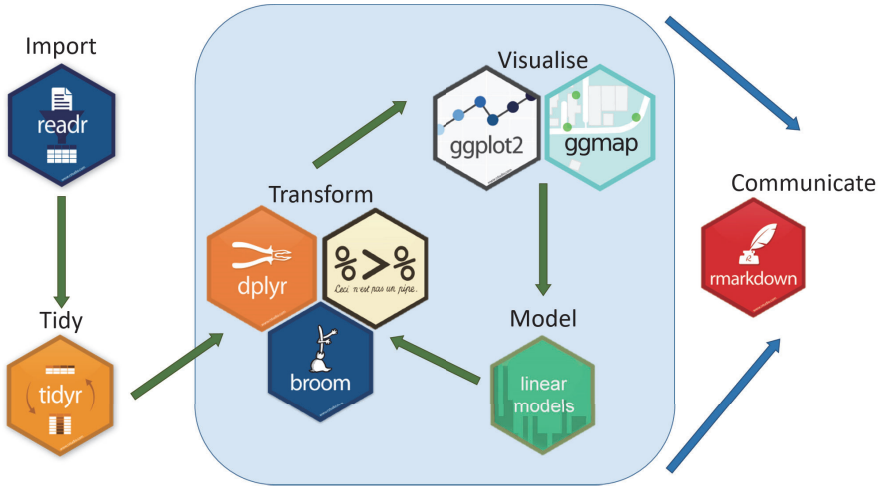


图 2: 数据科学的工作流程

即数据导入、数据清洗、数据变换、数据可视化、数据建模以及文档沟通，整个分析和探索过程，我们应当训练这样的数据思维。

0.2.2 什么是 R 语言？

1992 年，新西兰奥克兰大学统计学教授 Ross Ihaka 和 Robert Gentleman，为了方便地给学生教授统计学课程，他们设计开发了 R 语言（他们名字的首字母都是 R）。

- R 重要事件：
  - 2000 年，R 1.0.0 发布
  - 2005 年，ggplot2 包（2018.8 - 2019.8 下载量超过 1.3 亿次）
  - 2016 年，Rstudio 公司推出 tidyverse 包（数据科学当前最新 R 包）
  - 2020 年，R 4.0.0 发布，目前 CRAN 上的宏包数量 16159 个 R 包

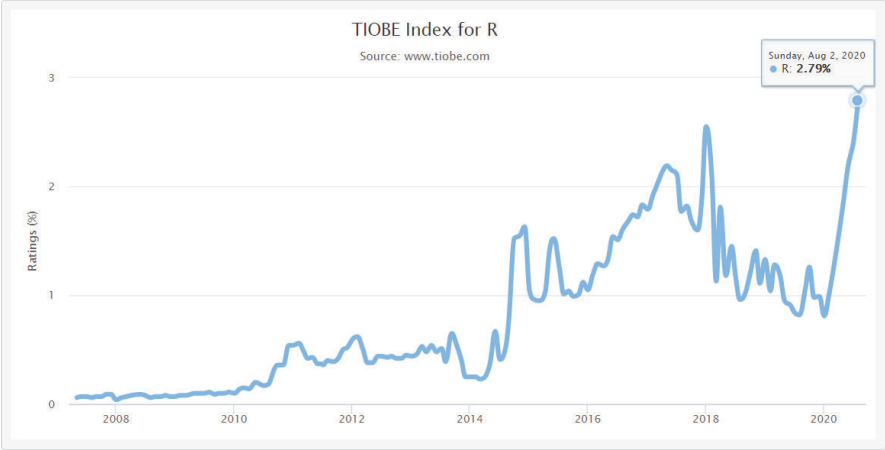


图 3: R 语言的 TIOBE 指数

Aug 2020	Aug 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.98%	+1.83%
2	1	↓	Java	14.43%	-1.60%
3	3		Python	9.69%	-0.33%
4	4		C++	6.84%	+0.78%
5	5		C#	4.68%	+0.83%
6	6		Visual Basic	4.66%	+0.97%
7	7		JavaScript	2.87%	+0.62%
8	20	⇈	R	2.79%	+1.97%
9	8	↓	PHP	2.24%	+0.17%
10	10		SQL	1.46%	-0.17%

图 4: TIOBE 最新编程语言排名

2019 年权威机构 KDnuggets 做过调研，显示数据科学领域最受欢迎的编程语言，包括 python 和 R：

- Python 更全能，适合将来做程序员或在工业企业工作
- R 更侧重数据统计分析，适合将来做科研学术

Software	2019 % share	2018 % share	2017 % share
Python	**65.8%**	65.6%	59.0%
RapidMiner	**51.2%**	52.7%	31.9%
R Language	**46.6%**	48.5%	56.6%
Excel	**34.8%**	39.1%	31.5%
Anaconda	**33.9%**	33.4%	24.3%
SQL Language	**32.8%**	39.6%	39.2%
Tensorflow	**31.7%**	29.9%	22.7%
Keras	**26.6%**	22.2%	10.7%
scikit-learn	**25.5%**	24.4%	21.9%
Tableau	**22.1%**	26.4%	21.8%
Apache Spark	**21.0%**	21.5%	25.5%

图 5: KDnuggets 数据科学编程语言排名

R 语言是用于统计分析，图形表示和报告的编程语言：

- R 语言是统计学家开发，为统计计算、数据分析和可视化而设计
- R 语言适合做数据处理和数据建模（数据预处理、数据探索性分析、识别数据隐含的模式、数据可视化）。

R 语言的优势：

- 免费开源，软件体积小根据需要安装扩展包，兼容各种常见操作系统，有强大活跃的社区

- 专门为统计和数据分析开发的语言，有丰富的扩展包
- 拥有顶尖水准的制图功能
- 面向对象和函数，比 Python 简单易学

在热门的机器学习领域：

- 有足以媲美 Python 的 sklearn 机器学习库的 R 机器学习包：`mlr3verse` 或 `tidymodels`

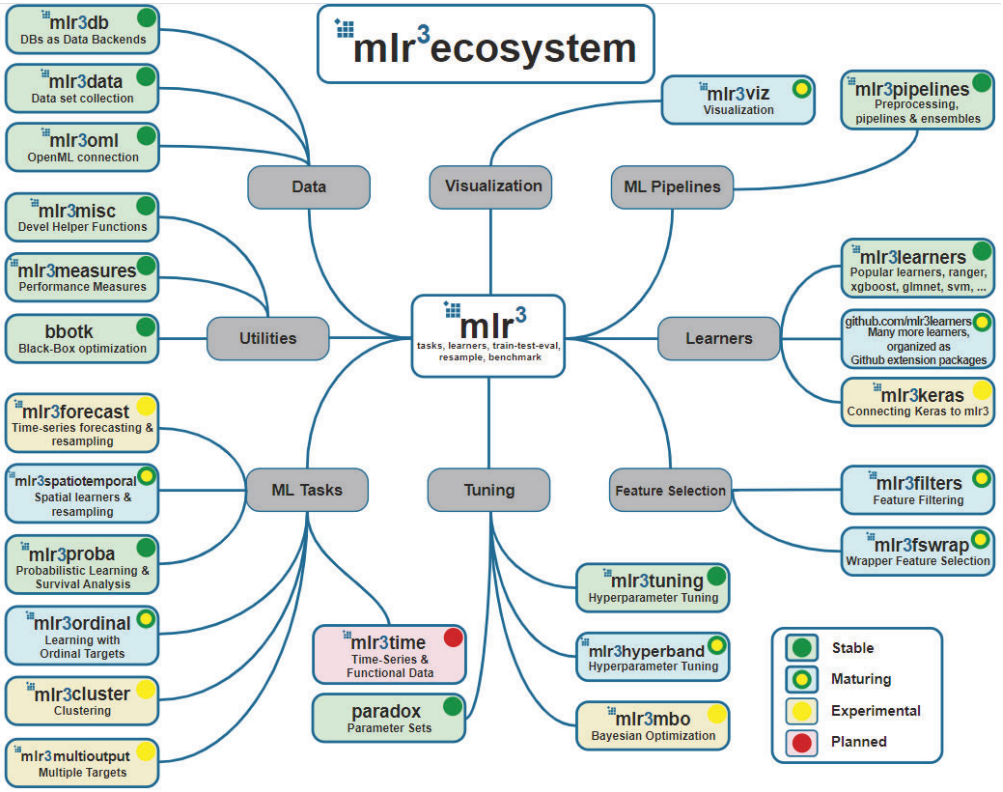


图 6: R 机器学习包 mlr3verse 的生态

本节部分内容参阅 (王敏杰2020).

0.2.3 一个改变了 R 的人

Hadley Wickham，R 语言超级大神，ggplot2, tidyverse 作者，RStudio 首席科学家，因对 R 长期以来的卓越贡献，荣获统计学界的诺贝尔奖：「**2019 COPSS 奖**」

图 7: R 语言大神 — Hadley Wickham

0.3 R 语言编程思想

0.3.1 面向对象

R 是一种基于对象的编程语言，即在定义类的基础上，创建与操作对象；数值向量、函数、图形等都是对象。Python 的**一切皆为对象**也适用于 R。

```
a = 1L
class(a)
```

```
## [1] "integer"
```

```
b = 1:10
class(b)
```

```
## [1] "integer"
```

```
f = function(x) x + 1
class(f)
```

```
## [1] "function"
```

早期和底层 R 语言中的面向对象编程是通过泛型函数来实现的，以 S3 类、S4 类为代表。新出现的 R6 类更适合用来实现通常所说的面向对象编程（OOP），包含类、属性、方法、继承、多态等概念。

面向对象的内容是 R 语言编程的高级内容，有上述基本了解即可，本书不做具体展开，只在附录放一个用 R6 类面向对象编程的简单示例，有兴趣的读者可参阅

(Wickham 2019a)。

0.3.2 面向函数

笼统来说，R 语言就两个事情：数据，对数据应用操作。这个操作就是函数，包括 R 自带的函数，各种扩展包里的函数，自定义的函数。

所以，使用 R 大部分时间都是在与函数打交道，学会了使用函数，R 语言也就学会了一半，啊？R 这么简单？？确实差不多[2]，很多人说 R 简单易学，也是因为此。

编程中的函数，是用来实现某个功能，其一般形式为：

```
(返回值 1, ..., 返回值 m) = 函数名 (输入 1, ..., 输入 n)
```

你只要把输入给它，它就能在内部进行相应处理，把你想要的返回值给你。

这些输入和返回值，在函数定义时，都要有固定的类型（模具）限制，叫做形参(形式上的参数)；在函数调用时，必须给它对应类型的具体数值，才能真正的去做处理，这叫做实参（实际的参数)。

所以，定义函数就好比创造一个模具，调用函数就好比用模具批量生成产品。

使用函数最大的好处，就是将实现某个功能，封装成模具，从而可以反复使用。这就避免了写大量重复的代码，程序的可读性也大大加强。

下面看一个调用函数的例子。比如想做线性回归，通过网络搜索知道是用自带 `lm()` 函数实现。那么先打开该函数的帮助：

```
?lm
```

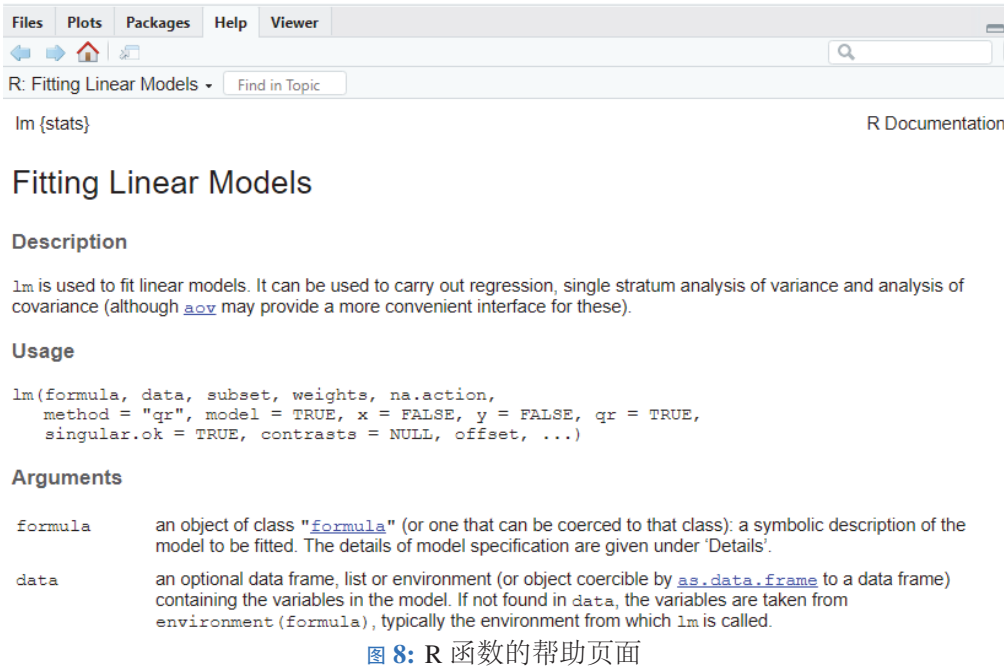


图 8: R 函数的帮助页面

[2]前提是不把 R 当一门编程语言，只想简单套用现成的算法模型。

执行 ? 函数名，若函数来自扩展包需要事先加载包，则在 Rstudio 右下角窗口打开函数帮助界面，一般至少包括如下内容：

- 函数描述
- 函数语法格式
- 函数参数说明
- 函数返回值
- 函数示例

通过阅读函数描述、参数说明、返回值，再调试示例，就能快速掌握该函数的使用。

函数包含很多参数，常用参数往往只是前几个。比如 `lm()` 的常用参数是：

- formula: 设置线性回归公式形式：因变量 ~ 自变量 + 自变量
- data: 提供数据（框）

使用自带的 `mtcars` 数据集演示，按照函数参数要求的对象类型提供实参：

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
model = lm(mpg ~ disp, data = mtcars)
summary(model)      # 查看回归汇总结果
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8922 -2.2022 -0.9631  1.6272  7.2305
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 29.599855   1.229720  24.070  < 2e-16 ***
## disp        -0.041215   0.004712  -8.747 9.38e-10 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.251 on 30 degrees of freedom
## Multiple R-squared:  0.7183,Adjusted R-squared:  0.709
## F-statistic: 76.51 on 1 and 30 DF,  p-value: 9.38e-10
```

所有的 R 函数，即使是陌生的，都可以这样来使用。

0.3.3 向量化编程

高级编程语言都提倡**向量化编程**[3]，说白了就是，对一列/矩阵/多维数组的数同时做同样的操作，既提升程序效率又大大简化代码。

向量化编程，关键是要用整体考量的思维来思考、来表示运算，这需要用到《线性代数》的知识，其实我觉得《线性代数》最有用的知识就是向量/矩阵化表示运算。

比如考虑 n 元一次线性方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \quad\cdots\cdots \qquad\quad \cdots\cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

若从整体的角度来考量，引入矩阵和向量：

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

则前面的 n 元一次线性方程组，可以向量化表示为：

$$Ax = b$$

可见，向量化表示大大简化了表达式。这放在编程中，就相当于本来用两层 `for` 循环才能表示的代码，简化为了短短一行代码。

向量化编程其实并不难，关键是要转变思维惯式：很多人学完 C 语言后的后遗症，就是首先想到的总是逐元素的 `for` 循环。摆脱这种想法，调动头脑里的《线性代数》知识，尝试用向量/矩阵表示，长此以往，向量化编程思维就有了。

下面以计算决策树算法中的样本经验熵为例来演示向量化编程。

[3]向量化编程中的向量，泛指向量/矩阵/多维数组。

对于数据集 D，$\frac{|D_k|}{|D|}$ 表示第 k 类样本所占的比例，则 D 的样本经验熵为

$$H(D) = -\sum_{k=1}^{K} \frac{|D_k|}{|D|} \ln \frac{|D_k|}{|D|}$$

其中，$|\cdot|$ 表示集合包含的元素个数。

实际中经常遇到要把数学式子变成代码，与前文自己写代码所谈到的一样，首先你要看懂式子，拿简单实例逐代码片段调试就能解决。

这里 D 是数据集，比如西瓜分类数据：

```
df = readxl::read_xlsx("datas/watermelon.xlsx")
knitr::kable(df)
```

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

因变量**好瓜**表示是否为好瓜，取值为“是”和“否”，则 D 分为两类：D_1 为好瓜类，D_2 为坏瓜类。因为只需要对各类做计数，只关心这一列数据就够了。

从内到外先要计算 $|D_k|/|D|, k = 1, 2$，用向量化的思维同时计算，就是统计各分类的样本数，再除以总样本数：

```
Y = df$好瓜
table(Y)                    # 计算各分类的样本数，得到向量
```

```
## Y
##  否  是
##   9   8
```

```
p = table(Y) / length(Y)    # 向量除以标量
p
```

```
## Y
##        否        是
## 0.5294118 0.4705882
```

继续代入公式计算，记住 R 自带函数天然就接受向量做输入参数：

```
log2(p)                     # 向量取对数
```

```
## Y
##         否         是
## -0.9175378 -1.0874628
```

```
p * log2(p)                 # 向量乘以向量，对应元素做乘法
```

```
## Y
##         否         是
## -0.4857553 -0.5117472
```

```
- sum(p * log2(p))          # 向量求和
```

```
## [1] 0.9975025
```

看着挺复杂的公式用向量化编程，核心代码只有两行：计算 p 和最后一行。这个实例虽然简单，但基本涉及到所有常用的向量化操作：

- 向量与标量做运算
- 向量与向量做四则运算
- 函数作用到向量

0.4 本书的特色与内容安排

0.4.1 本书的特色

1. 新

采用最新的 R 语言技术，甚至 R 包都用最新版本。比如三个月前迎来大版本更新的 dplyr 1.0.0，引入了 `across()` 函数代替数据操作函数的 `*_if`, `*_at`, `*_all` 后缀。并且，本书的电子版也将一直及时更新下去。

2. **试图讲透编程语法**

很多国内 R 语言编程书只是罗列堆砌编程语法，国外有不少优秀的 R 语言编程书，但翻译版往往就只是“直译”，只把表面意思用生硬的汉语表达出来，很难让初学者学透它们。

我写东西的特点就是，每个知识点都搜集很多相关最新资料，自己先学得透彻明白，再把自己的理解尽量简洁直白地表达出来。看过我知乎专栏文章或前面的引言的人，应当对此有所体会。

3. **精心准备实例**

编程语法讲透彻还不够，必须配以合适的实例来演示，所以也请读者一定要将编程语法讲解与配套实例结合起来阅读，比起调试实例代码，更重要的是借助实例代码理解透彻该编程语法。

4. **程序代码优雅、简洁高效**

本书程序代码都是基于最新的 `tidyverse`，自然就很优雅；简洁高效是因为我能用**向量化编程**就不用逐元素，能用**泛函式编程**，就不用 `for` 循环。

可以说，读者如果用我这本书入门 R 语言，或者更新你的 R 知识，就会自动跳过写低级啰嗦代码的阶段，直接进入写让别人羡慕的“高手级”代码的行列。

0.4.2 本书的内容安排

第一章先来讲述 R 语言编程的基本语法，这些语法在其它编程语言中也是相通的，包括搭建 R 语言环境、常用数据结构（存放数据的容器）、分支/循环结构、自定义函数。

第二章正式进入 `tidyverse` 流的数据操作，包括数据读写、数据连接、常用数据操作、数据清洗。

第三章是数据可视化，主要讲解 `ggplot2` 绘图语法、数据探索性分析、简单数据建模。

第四章讨论将 R 语言应用到常见统计分析、线性回归建模。

第五章是 R 语言的文档沟通，将简单讨论如何用 `Rmarkdown` 家族生成各种文档、书籍，`Shiny` Web 交互以及与 `Github` 沟通。