知乎用户对特朗普和希拉里的看法——基于爬虫和文本挖掘技术的分析

Ye Ling

## 摘要

学了R语言后就一直手痒，想从浩瀚的网络中抓一些数据来练练手。最近学习了爬虫终于可以一试身手了。

我写了一个专爬知乎内容的爬虫，只需输入知乎话题编号（如19552910，对应php语言)，就可以抓取该话题下所有的精华问题，以及每个问题下排名前十的最佳回答，包括回答的全部正文，作者，发布日期，点赞数和评论数。

本文从知乎上分别抓取了讨论美国大选的两位主角特朗普和希拉里的所有精华问答，共有18000多篇，并用文本挖掘技术加以分析。让我们看看知乎用户对特朗普和希拉里是怎么看的吧。有种观点认为知乎对特朗普是一边倒的支持，数据是否支持这种看法呢？

## 爬虫的编写

本文爬虫主要调用了rvest包，rvest是抓取静态网页的利器，使用起来远比RCurl包便捷。当然缺点是对动态加载的网页就无能为力了。

rvest最主要的三个函数是read\_html()，html\_nodes和html\_text()。功能如下：

read\_html()：传入网址和编码方式（中文网页一般是UTF-8），即可读取整个网页。 html\_nodes()：传入读取的网页和css节点，定位到所有符合要求的节点位置。 html\_text()：传入html\_nodes()的定位结果，提取所有节点下的文本内容，以向量的形式保存。

完整的爬虫程序如下：

library(rvest)  
library(magrittr) # 管道运算符  
  
# 输出一个数字，表示该话题的精华有多少页，需要传入话题编号  
get\_zhihu\_page <- function(topic) {   
 paste0("https://www.zhihu.com/topic/", topic, "/top-answers") %>%  
 read\_html(encoding="UTF-8") %>%  
 html\_nodes(".zm-invite-pager > span> a:nth-child(1)") %>%  
 html\_text() %>%  
 extract(3) %>%  
 as.numeric()  
}  
#get\_zhihu\_page(20023724) # 结果正确  
#get\_zhihu\_page(19674181)  
  
# 获取话题下第n页所有问题的网址  
# 第一页和其它页网址不一样，分别处理  
page2url <- function(n, topic) {  
 if (n==1) {  
 qstn <- paste0("https://www.zhihu.com/topic/", topic, "/top-answers") %>%  
 read\_html(encoding="UTF-8") %>%  
 html\_nodes("div.feed-item > div:nth-child(6) > div:nth-child(1) > h2:nth-child(1) > a:nth-child(1)") %>%  
 html\_attr("href") %>% # 获取全文链接  
 extract(grep("/question/", .)) #筛选出有用的链接  
 }  
 else {  
 qstn <- paste0("https://www.zhihu.com/topic/", topic, "/top-answers?page=", n) %>%  
 read\_html(encoding="UTF-8") %>%  
 html\_nodes("div.feed-item > div:nth-child(6) > div:nth-child(1) > h2:nth-child(1) > a:nth-child(1)") %>%  
 html\_attr("href") %>% # 获取全文链接  
 extract(grep("/question/", .)) #筛选出有用的链接  
 }  
 sub("http://www.zhihu.com", "", qstn) %>%  
 paste0("https://www.zhihu.com", .)  
}  
  
  
# 该话题下所有精华问题的网址  
get\_zhihu\_qstn <- function(topic) {   
 get\_zhihu\_page(topic) %>%  
 c(1:.) %>%  
 sapply(page2url, topic) %>%  
 unlist  
}  
#test <- get\_zhihu\_qstn(19674181) # 结果正确  
#get\_zhihu\_qstn(20023724) # 测试成功  
  
# 从问题的网址获取最热门的十条回答，组成data.frame形式  
# css节点的获取：用火狐浏览器打开相应的网页，鼠标停在需要抓取的内容的位置，右键查看元素，即跳至查看器相应位置，  
# 在该位置点右键，选复制-> css选择器即可。  
# 再复制几个同类型内容的css节点，观察异同，保留相同的部分，删去不同的部分（从冒号开始，包括冒号，一直删到">"前）。  
# 这样的节点就对应了这一类内容的所有内容  
qstn2ans <- function(url) {  
 web <- read\_html(url, encoding="UTF-8")  
   
 question <- web %>%  
 html\_nodes("span.zm-editable-content") %>%  
 html\_text()  
   
 author <- web %>%  
 html\_nodes("div.zm-item-answer > div:nth-child(6) > div:nth-child(1) > span") %>%  
 html\_text() %>%  
 tm::stripWhitespace()  
   
 text <- web %>%  
 html\_nodes("div.zm-item-answer > div:nth-child(7) > div:nth-child(2)") %>%   
 html\_text()  
   
 zan <- web %>%  
 html\_nodes("div.zm-item-answer > div:nth-child(5) > button:nth-child(1) > span:nth-child(2)") %>%  
 html\_text()  
   
 date <- web %>%  
 html\_nodes("div.zm-item-answer > div:nth-child(9) > div:nth-child(1) > a:nth-child(1)") %>%  
 html\_text()   
   
 commt\_count <- web %>%  
 html\_nodes("div.zm-item-answer > div:nth-child(9) > div:nth-child(1) > a:nth-child(2)") %>%  
 html\_text() %>%  
 tm::stripWhitespace()  
   
 data.frame(question=question, author=author,   
 text=text, zan=zan, date=date,   
 commt\_count=commt\_count, stringsAsFactors = F)  
}  
  
#tmp <- qstn2ans("https://www.zhihu.com/question/20594192")  
  
# 用户需要调用的函数  
# 传入话题编号，获得所有精华问答  
get\_zhihu\_ans <- function(topic) { # 输入话题编号  
 ans <- get\_zhihu\_qstn(topic) %>% # 得到所有问题网址  
 lapply(qstn2ans) %>%  
 Reduce(rbind, .)  
   
 write.csv(ans, paste0("E:/QUANT/text mining/ans\_of\_", topic, ".csv"))  
 return(ans)  
}

## 数据清洗

先来看看抓取的特朗普数据"ans\_of\_trp"和希拉里数据"ans\_of\_hlr"的结构。text字段就是回答的正文，我们主要分析这个字段。

# 爬取川普的数据  
#ans\_of\_trp <- get\_zhihu\_ans(20023724) # 耗时10分钟  
# 爬取希拉里的数据  
#ans\_of\_hlr <- get\_zhihu\_ans(19664274) # 耗时10分钟

#str(ans\_of\_trp)  
#str(ans\_of\_hlr)

对文本语料的数据清洗主要用到tm, qdap, Rwordseg, tmcn等几个包。过程包括：

1、自定义停用词库：文本中有很多词出现频繁，但对分析问题没有帮助，如“东西”，“方面”，“使用”等等，需要把这些干扰分析的词删除。虽然tmcn包已经提供了一个中文停用词库stopwordCN()，但还不够，不断地人工添加，这是一个繁琐又不可缺少的步骤。

2、自定义需要提取的词性：segmentCN()函数除了分词外，还提供每个词的词性，我们主要分析名词类词汇。

3、向词典中插入临时词汇：有些词分词词典中没有，又是我们预期会遇到的，如“邮件门”，这就需要手动插入。

4、语料清洗，包括去除数字，英文字符，标点符号，多余的空字符，分词，筛选出需要提取的词性，删除单个字的词，删除停用词。

5、最后生成Term Document Matrix(tdm)。

所用到的函数如下：

library(Rwordseg) # 中文分词  
library(tm)  
library(tmcn)  
library(wordcloud)  
library(wordcloud2)  
library(qdap)  
library(stringr)  
library(magrittr)  
library(plotrix) # 金字塔图  
library(dendextend) # 美化层次分类图  
library(RWeka)  
library(RColorBrewer)  
#loadDict()  
  
#VCorpus(list(language= NA)) # 因为语料是中英文混杂的，所以language=NA  
  
### 生成corpus前先对语料做预处理  
  
# 自定义停用词库  
myStopwords <- c(stopwordsCN(), "东西", "不会", "方面", "使用",  
 "需要", "没有", "觉得", "知道", "进行", "得到",  
 "应该", "出来", "部分", "用来", "实现", "适合",  
 "大的", "个人", "提供", "可能", "支持", "完成",  
 "感觉", "能够", "看到", "起来", "不能", "希望",   
 "实际", "利用", "回答", "包括", "例子", "最好",  
 "事情", "不用", "方向", "语言", "一行", "一部分",  
 "一回事", "什么的", "作者", "一句话", "不仅仅是",  
 "意思", "并不是", "另一个", "大多数人", "类似的",  
 "特别是", "越来越多", "不一定", "不可能", "有没有",  
 "特别是", "会不会", "有一天", "有一些", "这就是",  
 "不得不", "看起来", "取决于", "也就是", "能不能",  
 "相当于", "希拉里", "特朗普", "民主党", "共和党",  
 "奥巴马", "这些人", "候选人", "唐纳德", "尤其是",  
 "到时候", "意识到", "每个人")   
  
# 自定义需要的词性，乱码表示搜狗标准词库  
myNature <- c("n","vn","ns","鎼滅嫍鏍囧噯璇嶅簱", "userDefine")   
  
# 插入临时词汇  
ins\_words <- c("特朗普", "川普", "川粉", "川黑", "希粉", "希黑",  
 "希拉里", "维基解密", "阿桑奇", "克林顿", "奥巴马",  
 "政治正确", "白左", "伊万卡", "脱欧", "邮件门")  
insertWords(ins\_words, save = T)  
  
# 生成tdm需要的分词器，限定长度为1  
tokenizer <- function(x){  
 RWeka::NGramTokenizer(x, control = Weka\_control(min=1, max=1))  
}  
  
# 将原始语料转化为向量，每个回答为一个元素  
cre\_vector <- function(data) {  
 data %<>% extract(, "text") %>%  
 str\_replace\_all("[A-Za-z0-9]", "") %>%  
 removePunctuation() %>%  
 stripWhitespace() %>%   
 segmentCN(nature=T) %>% # 分词，并保留词性  
 lapply(function(vec) {   
 vec %<>% extract(names(.) %in% myNature)  
 }) %>% # 筛选名词和词组  
 lapply(function(vec) {   
 vec %<>% extract(nchar(.) >1) # 删去单个字的词汇  
 }) %>%  
 lapply(function(vec) {  
 vec %<>% removeWords(myStopwords)  
 }) %>%  
 lapply(function(vec) {  
 vec %<>% paste(collapse=" ")  
 }) %>% # 把所有的词重新粘在一起，以空格分割，形式上类似英文文本   
 unlist()  
}  
  
# 从vec生成tdm  
vec2tdm <- function(vec) {  
 vec %>% VectorSource() %>%  
 VCorpus(list(language= NA)) %>%  
 TermDocumentMatrix(list(tokenize=tokenizer))  
}

#trp\_vector <- cre\_vector(ans\_of\_trp)  
#trp\_tdm <- vec2tdm(trp\_vector)  
#hlr\_vector <- cre\_vector(ans\_of\_hlr)  
#hlr\_tdm <- vec2tdm(hlr\_vector)

## Including Code

You can include R code in the document as follows:

summary(cars)

## speed dist   
## Min. : 4.0 Min. : 2   
## 1st Qu.:12.0 1st Qu.: 26   
## Median :15.0 Median : 36   
## Mean :15.4 Mean : 43   
## 3rd Qu.:19.0 3rd Qu.: 56   
## Max. :25.0 Max. :120

## Including Plots

You can also embed plots, for example:



Note that the echo = FALSE parameter was added to the code chunk to prevent printing of the R code that generated the plot.