未知詞處理

Sung-Chien Lin

2018 年 9 月 25 日

課程簡介

課程內容

- 處理中文文本中的未知詞與了解中文詞的特性

學習目標

- 能夠運用各種統計訊息和經驗法則偵測文本中出現的未知詞
  - 字串出現的次數
  - 字串的左右複雜度
  - 字串的首尾字
  - 字串的組成
- 能夠說明文本處理中經常運用的 NGram 語言模型的意義

未知詞

- 大多數的時候中文處理需要先經過斷詞，斷詞的結果取決於好的詞典，也就是收錄完善的詞典。
- 然而每個領域都有這個領域專屬的詞，不僅如此，每天都有許多前所未見的詞出現在生活中。事實上，詞典無法收錄所有可能出現的詞。
- 斷詞系統無法確認詞典沒有收錄的詞，這些詞稱為未知詞(unknown words)。
- 為了使段詞的結果更好，除了以人工盡可能地收錄更多的詞到詞典中，另一的可能的做法是從文本中自動偵測未知詞，然後將偵測到的未知詞加入詞典。

未知詞偵測

- 未知詞的偵測一般來說有兩種，一種是根據文法規則，另一種設立用統計訊息與經驗法則。本次課程主要採用後者。
- 以下將利用這利用統計訊息與經驗法則偵測資料內的未知詞：
  - 字串出現次數
  - 字串的左右複雜度
  - 字串的首尾字
  - 字串的組成

NGram 相連詞模型

- 文本中接連出現的 N 個詞
- 相連的兩個詞稱為 Bigram，三個詞稱為 Trigram
- 本次課程利用 NGram 找出文本中所有接連出現的 N 個詞
  - 計算候選詞語出現次數
  - 以(N+1)Gram 估計 NGram 的左右複雜度，判斷 NGram 是否可能是詞語

本次課程需先安裝套件

- 本次課程需要使用 tidytext 套件，產生 NGram 資料，並且計算 NGram 在文件中的出現次數。
- 請在 Console 上輸入

```
install.packages("tidytext")
```

本次課程程式

```
setwd("rCourse/08")

library(readr)
library(tidyverse)
library(jiebaR)
library(tidytext)

news <- data.frame()

# 讀入要處理的新聞資料
for (i in 1:15) {
  date <- sprintf("2018_09_%02d", i)

  news.df <- read_csv(file=paste0("udn_", date,".csv"))
  news <- rbind(news, news.df) #將讀入的新聞資料與原先的資料合併
}

# 設定斷詞系統
mp.seg <- worker(type="mp", symbol=TRUE)

# 進行斷詞
news <- news %>%
  filter(!is.na(text)) %>%  # 過濾空的新聞資料
  mutate(id=row_number()) %>%
  rowwise() %>%
  mutate(text.sg=paste0(segment(paste(title, text, sep="。"), mp.seg),
collapse=" ")) %>%
  ungroup()
```

```
# 統計輸入的文本內每一個可能的NGram 出現的次數，計算候選詞語出現的次數與左右
複雜度
NGramGenerator <- function(df, n) {
  # df 為輸入之文本, n 是NGram 的N
  this_ngram <- df %>%
    select(id, text.sg) %>%
    unnest_tokens(ngram, text.sg, token="ngrams", n=n) %>% # 利用tidyte
xt 套件提供的功能從文本產生NGram
    count(ngram, sort=TRUE) %>% # 統計每一個NGram 出現次數
    rename(c=n)

  return(this_ngram)
}

###########
# 根據頻率及字串的樣式(pattern)判斷是否為可能的候選詞語
CandidateSelector <- function(charstr, n) {
  # charstr 為可能的候選詞(即NGram)，n 為NGram 的N
  count.th <- 20

  lstr <- "^\\p{L}+"  # 全部為數字或文字組成的字串，若不是這種字串便需要排除
  estr <- "^[a-zA-Z]+" # 全部由英文字母組成的字串，須排除
  for (i in 1:(n-1)) {
    lstr <- paste0(lstr, "\\s\\p{L}+")
    estr <- paste0(estr, "\\s[a-zA-Z]+")
  }
  lstr <- paste0(lstr, "$")
  estr <- paste0(estr, "$")

  stopwords <- c("的", "在", "是", "都", "了", "也", "很", "會", "有", "
呢", "嗎", "就", "但", "所", "我", "不", "到", "要", "於")
```

```
  charstr <- charstr %>%
    filter(c>count.th) %>% # 取出總頻次大於count.th的Ngram
    filter(grepl(lstr, ngram, perl=TRUE)) %>% #保留內容為數字和文字的NGr
am(也就是去除包含符號的NGram)
    filter(!grepl(estr, ngram, perl=TRUE)) %>% # 排除全英文字母的Trigram
    filter(!(substr(ngram, 1, 1) %in% stopwords)) %>% # 去除第一字為停用
字的NGram
    filter(!(substr(ngram, nchar(ngram), nchar(ngram)) %in% stopwords))
 # 去除最後一字為停用字的NGram

  return(charstr)
}

########
# 取出NGram前面的(N-1)個相連詞
headString <- function (ngram, n) {
  sapply(ngram, function (x, n) {
    last_space = gregexpr(" ", x)[[1]][n-1]
    return(substr(x, 1, last_space-1))
  }, n)
}

# 取出NGram後面的(N-1)個相連詞
tailString <- function (ngram) {
  sapply(ngram, function (x) {
    first_space = gregexpr(" ", x)[[1]][1]
    return(substr(x, first_space+1, nchar(x)))
  })
}
```

```
# 依據左右複雜度判斷是否為可能的候選詞語，複雜度太小比較不可能是詞語
LRJointEstimator <- function(this_ngram, nextone, n) {
  # this_ngram 目前處理的NGram，nextone 為(N+1)Gram，n 為 NGram 的N
  complex.th <- 1

  # 計算右接的複雜度
  ngram_at_head <- nextone %>%
    mutate(headstr=as.character(headString(ngram, n+1))) %>% # 找出所有(N+1)Gram 的前N 個相連詞
    inner_join(this_ngram, by=c("headstr"="ngram")) %>% # 比對(N+1)Gram的前N 個相連詞與要判斷的NGram
    group_by(headstr) %>% # 利用熵(entropy)公式計算右接複雜度
    mutate(ratio=c.x/sum(c.x)) %>%
    mutate(ent=-log(ratio)*ratio) %>%
    summarise(rjoint=sum(ent))

  # 計算左接的複雜度
  ngram_at_tail <- nextone %>%
    mutate(tailstr=as.character(tailString(ngram))) %>% # 找出所有(N+1)Gram 的後N 個相連詞
    inner_join(this_ngram, by=c("tailstr"="ngram")) %>% # 比對(N+1)Gram的後N 個相連詞與要判斷的NGram
    group_by(tailstr) %>% # 利用熵(entropy)公式計算左接複雜度
    mutate(ratio=c.x/sum(c.x)) %>%
    mutate(ent=-log(ratio)*ratio) %>%
    summarise(ljoint=sum(ent))

  this_ngram <- this_ngram %>%
    left_join(ngram_at_head, by=c("ngram"="headstr")) %>% # 加上右接複雜度
    left_join(ngram_at_tail, by=c("ngram"="tailstr")) %>% # 加上左接複雜
```

```
度
    filter(ljoint>=complex.th) %>% # 過濾左右複雜度大小的字串
    filter(rjoint>=complex.th)

  return(this_ngram)
}
####

bigram <- NGramGenerator(news, 2) # 產生文本中所有 Bigram 及其出現次數

##
bigram <- CandidateSelector(bigram, 2) # 根據出現次數及字串的樣式(patter
n)判斷 Bigram 是否為可能的候選詞語

trigram <- NGramGenerator(news, 3) # 產生文本中所有 Trigram 及其出現次數

bigram <- LRJointEstimator(bigram, trigram, 2) # 依據左右複雜度判斷 Bigra
m 是否為可能的候選詞語

##
trigram <- CandidateSelector(trigram, 3) # 根據出現次數及字串的樣式(patte
rn)判斷 Trigram 是否為可能的候選詞語

fourgram <- NGramGenerator(news, 4) # 產生文本中所有 Fourgram 及其出現次數

trigram <- LRJointEstimator(trigram, fourgram, 3) # 依據左右複雜度判斷 Tr
igram 是否為可能的候選詞語

##
fourgram <- CandidateSelector(fourgram, 4) # 根據出現次數及字串的樣式(pat
```

```
tern)判斷 Fourgram 是否為可能的候選詞語

fivegram <- NGramGenerator(news, 5) # 產生文本中所有 Fivegram 及其出現次數

fourgram <- LRJointEstimator(fourgram, fivegram, 4) # 依據左右複雜度判斷
Fourgram 是否為可能的候選詞語

##################
# 合併所有候選詞語篩選結果
dict <- data.frame() %>%
  bind_rows(bigram) %>%
  bind_rows(trigram) %>%
  bind_rows(fourgram) %>%
  select(ngram)

## 去除字串中的空白
dict$ngram <- gsub("\\s", "", dict$ngram, perl=TRUE)

## 寫入檔案
write.table(dict, file="udn_2018_03.dict", quote=FALSE,
            row.names=FALSE, col.names=FALSE, fileEncoding="UTF-8",
            append=TRUE)
```

預備工作

準備工作目錄與檔案

- 在 rCourse 下，建立工作目錄 08
- 將 08 的 2018 年三月新聞資料的所有檔案複製到 09 下

設定工作目錄

- 首先開啟新的 Script
- 在 Script 上，設定工作目錄

```
setwd("rCourse/08")
```

載入此次課程所需套件

- 在 Script 上輸入

```
library(readr)
library(tidyverse)
library(jiebaR)
library(tidytext)
```

讀入文字資料並進行斷詞

讀入聯合報九月生活新聞資料

- rbind()：以 row 的方式，合併兩個 data frame
- 在 Script 上輸入

```
news <- data.frame()


for (i in 1:15) {
  date <- sprintf("2018_09_%02d", i)


  news.df <- read_csv(file=paste0("udn_", date,".csv"))
  news <- rbind(news, news.df) #將讀入的新聞資料與原先的資料合併
}
```

設定 jieba 斷詞器

- 斷詞器的參數
  - type="mp"使用最長比對法，不會產生詞典中沒有的詞
  - symbol=TRUE 輸出符號
- 在 Script 上輸入

```
mp.seg <- worker(type="mp", symbol=TRUE)
```

將文字資料斷詞

- 在 Script 上輸入

```
news <- news %>%
  filter(!is.na(text)) %>%  # 過濾空的新聞資料
  mutate(id=row_number()) %>%
  rowwise() %>%
  mutate(text.sg=paste0(segment(paste(title, text, sep="。") mp.seg), c
ollapse=" ")) %>% #詞語之間加入空白間隔，提供 NGram Tokenizer 處理
  ungroup()
```

- 請在 Console 輸入 View(news)，檢視文字資料與斷詞結果

	text	id	text.sg
342674	板橋麥當勞10點30分真的打烊了，門口擺滿了準備補貨的麵...	1	麥當勞 之 亂 10:30 之後 大部分 餐廳 暫停營業 。 板橋 麥...
343358	民眾搭計程車時，司機直接拿出一本厚厚的地圖找路，令她...	2	老 司機 拿出 超 厚 自 製 地圖 找 路 網友 感嘆 「 看 了 好想...
342739	麥當勞今天上午10時30分起，大部分餐廳暫停營業。記者王...	3	麥當勞 之 亂 ！ 員工 ： 累 得 跟 狗 一樣 睡覺 做 惡夢 。 麥...
343148	近期網路傳出「聞不到花生醬，就是得失智症」訊息，國健...	4	聞 花生醬 測 失 智 症 ？ 出現 這 3 種 警訊 更 應該 當心 。 ...
342651	昨麥當勞推出大麥克買一送一，民眾排隊到門外。記者鄭清...	5	麥當勞 之 亂 ？ 插隊 、 違停 、 塞車 「 員工 苦難 日 」 。 ...
342622	詐騙臉書頁面把球后戴資穎寫成戴「姿」穎。圖／擷取自臉...	6	小心 受騙 ！ 搭 大 麥克 熱潮 假 優惠 「 資 、 姿 」 不 分 。...
342639	強烈颱風燕子明日至下周一接近日本南方海域。圖／翻攝自...	7	吳 德 榮 ： 燕子 增強 為 今年 最 強颱風 下 周二 起 襲 日 。 ...

- 在新聞中的許多詞語因為沒有收錄在詞典，無法斷出，稱為未知詞(unknown words)

產生 NGram

BiGram (詞雙連)

- unnest_tokens 將欄位資料拆解成 tokens，然後進行 unnest
  - token="ngrams"：利用 ngrams 做為 tokenizer，將輸入的文字資料拆解成較小的文字單位(tokens)
  - n=2：設定產生的 tokens 為詞雙連(bigram)
  - unnest_tokens：將拆解開的 tokens 展開成觀測值
- count(ngram, sort=TRUE)
  - 計算各 ngram 的出現總頻次，儲存在名稱為 n 的欄位中
  - sort=TRUE，由大到小排序
- rename(c=n)：將 data frame (tibble)中的 n 欄位改名為 c
- 在 Script 上輸入

```
bigram <- news %>%
  select(id, text.sg) %>%
  unnest_tokens(ngram, text.sg, token="ngrams", n=2) %>% # 利用斷詞的結果，找出所有的ngram (目前n=2，也就是Bigram)
  count(ngram, sort=TRUE) %>% # 統計ngram次數
  rename(c=n)
```

ngram	c
a c	3522
o o	3494
b o	3482
c e	3467
f a	3395
e b	3357
o k	3334
分享 f	3241

TriGram (詞三連)

- 在 tibble trigram 內，ngram 為詞三連
- c 為該詞三連的出現總頻次
- 利用 trigram 估計 bigram 的左右複雜度
- trigram 也可以做為接下來判斷是否為詞語的候選字串

```
trigram <- news %>%
  select(id, text.sg) %>%
  unnest_tokens(ngram, text.sg, token="ngrams", n=3) %>%
  count(ngram, sort=TRUE) %>%
  rename(c=n)
```

ngram	c
b o o	3326
a c e	3315
o o k	3313
e b o	3286
f a c	3273
c e b	3262
分享 f a	3241
提供 分享 f	1720

將產生 NGram 的部分寫成函數的形式

- 我們可以觀察到上面兩段程式間有許多重複的地方，所以可以將它們改寫成函數，只要改變 n 的數值便可以產生我們需要的 NGram
- df 為輸入的文本
- n 是 NGram 的 N

```
NGramGenerator <- function(df, n) {
  this_ngram <- df %>%
    select(id, text.sg) %>%
    unnest_tokens(ngram, text.sg, token=ngram_tokens, n=n) %>%
    count(ngram, sort=TRUE) %>%
    rename(c=n)

  return(this_ngram)
}
```

利用出現次數、文字形式、停用詞等選擇可能的候選詞語

取出出現總次數大於 count.th 的 BiGram

- 出現太少次的 NGram 很有可能是偶然碰在一起的字串，我們認為出現總次數大於某一個閾值的 NGram 才比較有可能是一個詞
- 另一方面，雖然出現次數較少的字串仍然有可能是詞語，但以計算成本而言，這些出現次數較少的詞語可以省略。
- 在本次課程中，將出現總次數的閾值設為 20。
- 在 Script 上輸入

```
count.th <- 20


bigram <- bigram %>%
  filter(c>count.th)
```

取出內容為文字(去除內容中包含符號)的 BiGram

- 我們假定詞語當中的組成必須純粹是文字，不可能有符號，所以一旦 NGram 當中有任何的符號，我們便捨棄這個 NGram。
- 以下利用 Regular Expression 的方法判斷詞語是否純粹由文字組成。
- 以 Bigram 而言，為中間為空白的兩段文字。將 Bigram 寫成 Regular Expression："^\\p{L}+\\s\\p{L}+$"
  - ^ → 字串開頭
  - \\p{L} → 文字或數字(不包含符號)
  - \\p{L}+ → 一個或以上的文字或數字
  - \\s → 空白
  - $ → 字串結尾
- 在 Script 上輸入

```
bigram <- bigram %>%
  filter(grepl("^\\p{L}+\\s\\p{L}+$", ngram, perl=TRUE))
```

排除全英文字母的 Bigram

- 我們假定如果全部由英文字母組成的字串，不是我們要提取的詞語。
- 在 Script 上輸入
- "^[a-zA-Z]+\\s[a-zA-Z]+$"
  - [a-zA-Z] → 英文字母

```
bigram <- bigram %>%
  filter(!grepl("^[a-zA-Z]+\\s[a-zA-Z]+$", ngram, perl=TRUE))
```

去除第一字或最後一字為停用詞的 BiGram

- 首尾字是停用詞的字串，通常是在某一個詞語的前或後加上這個停用詞，所以將它們去除。
- 在 Script 上輸入

```
stopwords <- c("的", "在", "是", "都", "了", "也", "很", "會", "有", "呢", "嗎", "就", "但", "所", "我", "不", "到", "要", "於")


bigram <- bigram %>%
  filter(!(substr(ngram, 1, 1) %in% stopwords)) %>%
  filter(!(substr(ngram, nchar(ngram), nchar(ngram)) %in% stopwords))
```

Filter

ngram	c
分享 f	3241
台 灣	2167
提供 分享	1720
民 眾	1040
攝影 分享	746
運 分析	540
因 為	489
北 市	353

練習

- 將上述有關 bigram 篩選經驗法則的程式敘述利用 pipe(%>%)串接起來

將上述篩選經驗法則寫成函數的形式

```
CandidateSelector <- function(this_ngram, n) {
  count.th <- 20

  lstr <- "^\\p{L}+"
  estr <- "^[a-zA-Z]+"
  for (i in 1:(n-1)) {
    lstr <- paste0(lstr, "\\s\\p{L}+")
    estr <- paste0(estr, "\\s[a-zA-Z]+")
  }
  lstr <- paste0(lstr, "$")
  estr <- paste0(estr, "$")

  stopwords <- c("的", "在", "是", "都", "了", "也", "很", "會", "有", "
呢", "嗎", "就", "但", "所", "我", "不", "到", "要", "於")


  this_ngram <- this_ngram %>%
    filter(c>count.th) %>% # 取出總頻次大於count.th 的Ngram
    filter(grepl(lstr, ngram, perl=TRUE)) %>% #取出內容為文字(去除包含其
中包含符號)的NGram
    filter(!grepl(estr, ngram, perl=TRUE)) %>% # 排除全英文字母的Ngram
    filter(!(substr(ngram, 1, 1) %in% stopwords)) %>% # 去除第一字為停用
字的NGram
    filter(!(substr(ngram, nchar(ngram), nchar(ngram)) %in% stopwords))
 # 去除最後一字為停用字的NGram

  return(this_ngram)
}
```

利用左右複雜度判斷詞語間隔

利用左右複雜度評估候選字串是否為詞語的可能性

- 如果候選字串是一個詞語，其左右便是另外的詞語字串。
- 詞語的左右能夠接的詞語字串種類必然相當多，而且很難事先預測。
- 反之，如果候選字串的左或右字串，種類相當少而容易預測，便可能不是一個詞
  - 例如：「書館」的左邊，通常會是什麼呢？
  - 又如：「立法委」的右邊，通常又會是什麼呢？
- 因此，我們可以利用左右複雜度評估候選字串是否為詞語的可能性
  - 候選字串的左右複雜度都很大時便可能是一個詞語

如何知道候選字串的左邊曾出現過那些詞語

- 利用(N+1)Gram
- 將(N+1)Gram 的後面 N 個詞視為是候選字串，(N+1)Gram 的第一個詞便是這個候選字串左邊曾經出現過的詞語之一
- 因此，只要將文件集合所有曾經出現過的(N+1)Gram，取出後面 N 個詞，加以分群。使得分群後，每一群的(N+1)Gram 後面 N 個詞都相同。從每一群的(N+1)Gram，其所有的第一個詞，便可以知道候選字串的左邊曾出現過那些詞語。

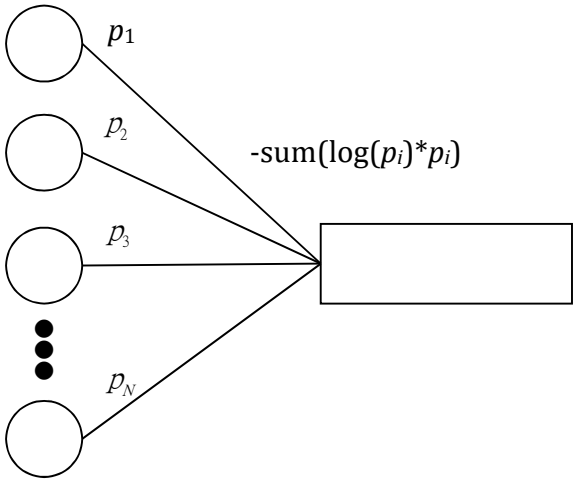
從(N+1)Gram 中取出後面的 N 個詞語

- 找出第一個空白，first_space 是這個空白的位置
- 從字串中取出第一個空白後一個字到最後一個字之間的部分字串

```
tailString <- function (ngram) {
  sapply(ngram, function (x) {
    first_space = gregexpr(" ", x)[[1]][1]
    return(substr(x, first_space+1, nchar(x)))
  })
}
```

利用 Shannon Entropy 的概念計算字串的左右複雜度

- entropy = -sum(log(p_i)*p_i)
- p_i： 事件的頻率(在此為字串左邊出現某一詞語的頻率)
- entropy 愈大，該字串可能的左接詞語愈複雜
  - 左接詞語的種類愈多
  - 分布在各左接詞語愈平均(愈難預測是哪一個詞語)



利用 Shannon Entropy 的概念計算字串的左接複雜度

```
bigram_at_tail <- trigram %>%
  mutate(tailstr=as.character(tailString(ngram))) %>% #找出所有 Trigram 中後面兩個詞語
  inner_join(bigram, by=c("tailstr"="ngram")) %>% #與 Bigram 中的候選詞語比對
  group_by(tailstr) %>%  #進行 Entropy 計算，做為左接複雜度
  mutate(ratio=c.x/sum(c.x)) %>%
  mutate(ent=-log(ratio)*ratio) %>%
  summarise(ljoint=sum(ent))
```

tailstr	ljoint
a 提供	1.6150686
e 分享	0.4305750
e 系列	1.4898688
e 提供	2.0633139
g 分享	0.0000000
h 提供	0.6870920
i 提供	1.6220340
k 中央	0.0000000

- 左接複雜度較大，候選字串的左邊有較多種類的詞語，代表候選字串可能是某一詞語的開頭。
- 左接複雜度較小，代表候選字串的左邊曾經出現的詞語種類有限，候選字串比較可能是詞語右邊的一部份。因此，比較不可能是完整的詞語。

如何知道候選字串的右邊曾出現過那些詞語

- 將(N+1)Gram 的前面 N 個詞視為是候選字串，(N+1)Gram 的最後一個詞便是這個候選字串右邊曾經出現過的詞語之一
- 因此，只要將文件集合所有曾經出現過的(N+1)Gram，取出前面 N 個詞，加以分群。使得分群後，每一群的(N+1)Gram 前面 N 個詞都相同。從每一群的(N+1)Gram，其所有的最後一個詞，便可以知道候選字串的右邊曾出現過那些詞語。

從(N+1)Gram 中取出前面的 N 個詞語

- 找出最後一個空白，last_space 是這個空白的位置
- 從字串中取出第 1 個字到最後一個空白前一個字之間的部分字串

```
headString <- function (ngram, n) {
  sapply(ngram, function (x, n) {
    last_space = gregexpr(" ", x)[[1]][n-1]
    return(substr(x, 1, last_space-1))
  }, n)
}
```

利用 Shannon Entropy 的概念計算字串的右接複雜度

```
bigram_at_head <- trigram %>%
  mutate(headstr=as.character(headString(ngram, 3))) %>% #找出所有Trigram 中前面兩個詞語
  inner_join(bigram, by=c("headstr"="ngram")) %>% #與Bigram 中的候選詞語比對
  group_by(headstr) %>%    #進行Entropy 計算，做為右接複雜度
  mutate(ratio=c.x/sum(c.x)) %>%
  mutate(ent=-log(ratio)*ratio) %>%
  summarise(rjoint=sum(ent))
```

headstr	rjoint
a 提供	0.00000000
e 分享	0.00000000
e 系列	2.44976768
e 提供	0.00000000
g 分享	0.00000000
h 提供	0.11924693
i 提供	0.08256494
k 中央	0.50104691

將 bigram 上的候選字串加上左右複雜度

```
bigram <- bigram %>%
  left_join(bigram_at_head, by=c("ngram"="headstr")) %>%
  left_join(bigram_at_tail, by=c("ngram"="tailstr"))
```

ngram	c	rjoint	ljoint
分享 f	3241	0.0000000	1.9217757
台 灣	2167	5.8590364	5.7538825
提供 分享	1720	0.0000000	4.7490108
民 眾	1040	5.5621051	5.0337068
攝影 分享	746	0.0000000	4.2006092
運 分析	540	4.3939799	1.0986123
因 為	489	5.6506778	5.0603944
北 市	353	4.3342313	1.5219967

- 當 NGram 的左接複雜度或右接複雜度較小時，其便比較不可能為一個詞語，例如：「分享 f」的右接複雜度只有 0，右邊能接的詞語數只有一個，因此「分享 f」不會是一個完整的詞語；而「台 灣」的左右複雜度都相當大，很明顯的左右都會接許多種類的詞語，所以「台 灣」是一個完整的詞語。

刪除左右複雜度小於 complex.th 的字串

```
complex.th <- 1


bigram <- bigram %>%
  filter(ljoint>=complex.th) %>%
  filter(rjoint>=complex.th)
```

ngram	c	rjoint	ljoint
台灣	2167	5.859036	5.753882
民眾	1040	5.562105	5.033707
運分析	540	4.393980	1.098612
因為	489	5.650678	5.060394
北市	353	4.334231	1.521997
新北	339	1.291641	4.808086
亞培	336	3.649137	4.561538
高雄	327	4.518345	4.521820

將左右複雜度的過濾經驗法則寫成函數的形式

```
LRJointEstimator <- function(this_ngram, nextone, n) {
  complex.th <- 1

  ngram_at_head <- nextone %>%
    mutate(headstr=as.character(headString(ngram, n+1))) %>%
    inner_join(this_ngram, by=c("headstr"="ngram")) %>%
    group_by(headstr) %>%
    mutate(ratio=c.x/sum(c.x)) %>%
    mutate(ent=-log(ratio)*ratio) %>%
    summarise(rjoint=sum(ent))

  ngram_at_tail <- nextone %>%
    mutate(tailstr=as.character(tailString(ngram))) %>%
    inner_join(this_ngram, by=c("tailstr"="ngram")) %>%
    group_by(tailstr) %>%
    mutate(ratio=c.x/sum(c.x)) %>%
    mutate(ent=-log(ratio)*ratio) %>%
    summarise(ljoint=sum(ent))

  this_ngram <- this_ngram %>%
    left_join(ngram_at_head, by=c("ngram"="headstr")) %>%
    left_join(ngram_at_tail, by=c("ngram"="tailstr")) %>%
    filter(ljoint>=complex.th) %>%
    filter(rjoint>=complex.th)

  return(this_ngram)
}
```

本次課程小結

小結

- 本次課程利用字串的出現次數、字串的首尾字、字串的組成、字串的左右複雜度等統計訊息和經驗法則發現可能的未知詞
- 從文本發現未知詞是重複性很高的工作
- 可設法利用迴圈與函數等程式設計完成這些工作

延伸思考

1. 在本次課程中利用統計訊息和經驗法則偵測未知詞。雖然是依據電腦的演算法自動化判斷候選字串是否為未知詞，但分析人員仍然必須決定許多參數，例如字串出現次數的高低和左右複雜度的閾值(threshold)。當閾值設定得較高，篩選進來的字串數較少，可能是未知詞的機會愈大，但有可能遺漏許多出現次數較低或左右複雜度較低的未知詞。反之，閾值設定得較低，篩選進來的字串數較多，雖然可能找到更多的未知詞，但也可能帶入許多錯誤。因此，如何決定這些參數對分析的結果相當重要。請你嘗試看看調整不同的參數，找出較好的結果。

2. 除了本次課程中所利用字串出現總次數、字串的組成、字串的首尾字和字串的左右複雜度等統計訊息和經驗法則之外，請問你是否可以設計出利用其他的統計訊息和經驗法則來進行偵測的方法，提高偵測的效能？

3. 在偵測出未知詞之後，我們將它們可以加入詞典，重新進行斷詞，並且利用上次課程的關鍵詞語擷取方法，選出文件集合相關的關鍵詞語。現在請你思考看看如何在研究中應用某一個特定文件集合相關的關鍵詞語？