自然言語処理入門

岸山 健 (31-187002)

Dec. 10, 2018

課題

「ヒロシが病院でもらった薬を飲んだ」を CYK 法で解析せよ.

1　概要と回答

CYK(Cocke–Younger–Kasami) 法は最も基本的なボトムアップアルゴリズムの一つである． 行列 a の対角成分に前終端記号 (つまり品詞 (POS)) を配置し，3 重の for 文で解析を行っていく.

```
# ヒロシ　 が　 病院　 で　 もらった　 薬　 を　 飲んだ # <- 終端記号
#　 "NP"　 "P"　 "NP"　 "P"　　 "VP" "NP"　 "P"　　 "VP" # <- 前終端記号
```

品詞列の長さを d とすると， 解析が終わった時点での $a_{1,d}$ が持つ状態で 文が適格か不適格かを判断できる. この CYK 法を用いて与えられた文字列と文法に統語解析を行なうと，以下の表が得られる．非終端記号が 2 つ以上ある箇所に構造的な曖昧性が存在する.

```
#      [,1] [,2] [,3] [,4] [,5]    [,6] [,7] [,8]
# [1,] "NP" "PP" ""   ""   "S, VP" "NP" "PP" "S, VP, S, VP, S, VP"
# [2,] ""   "P"  ""   ""   ""      ""   ""   ""
# [3,] ""   ""   "NP" "PP" "S, VP" "NP" "PP" "S, VP, S, VP"
# [4,] ""   ""   ""   "P"  ""      ""   ""   ""
# [5,] ""   ""   ""   ""   "VP"    "NP" "PP" "S, VP"
# [6,] ""   ""   ""   ""   ""      "NP" "PP" "S, VP"
# [7,] ""   ""   ""   ""   ""      ""   "P"  ""
# [8,] ""   ""   ""   ""   ""      ""   ""   "VP"
```

一番右上 $a_{1,8}$ のレベルでは S が 3 つ作成されているため，解析の過程で 3 つの曖昧性が発生したと分かる. まず $a_{1,8}$ に S を返した組み合わせを探すと，$a_{1,2}$ の PP「ヒロシが」と $a_{3,8}$ の VP「病院でもらった薬を飲んだ」 が見つかる． 今回の場合 $a_{3,8}$ に VP が 2 つあるため， 曖昧性は $a_{1,2}$ の PP「ヒロシが」ではなく $a_{3,8}$ の VP で発生している.

そこで i が 3 から始まる PP($a_{3,k}$) と j が 8 で終わる VP($a_{k+1,8}$) のペアを探すと，「病院でもらった薬を ($a_{3,7}$)」「飲んだ ($a_{8,8}$)」と「病院で ($a_{3,4}$)」「もらった薬を飲んだ ($a_{5,8}$)」のパターンが該当する．したがって以下の構造的曖昧性があり，句構造は以下の 2 つがある.

- $[_S\ [_{PP}$ ヒロシが$]\ [_{VP}\ [_{PP}$ 病院でもらった薬を$]$ 飲んだ$]]$
- $[_S\ [_{PP}$ ヒロシが$]\ [_{VP}[_{PP}$ 病院で$]$ もらった薬を飲んだ$]]$

他方，$a_{1,7}$「ヒロシが病院でもらった薬を (PP)」と $a_{8,8}$「飲んだ (VP)」の組み合わせもある．こちらは構造的な曖昧性が発生しておらず，「病院で」という PP と「もらったという」VP が マージして「病院でもらった」という VP となり，さらに「ヒロシが」という PP と 「病院でもらった」という VP がマージして「ヒロシが病院でもらった」という VP が作られている．この VP と「薬」がマージし NP，そしてさらに 「を」とマージして PP となる．これは「ヒロシが病院でもらってきた薬」を誰かは知らないけど誰かが飲んだ，という意味になる．

したがって，得られるく構造は上の 2 つに以下を加えたものである．曖昧性が生じない PP より下のレベルは省略した．

- $[_S\ [_{PP}\ [_{NP}\ [_{VP}\ [_{PP}$ ヒロシが$]\ [_{VP}\ [_{PP}$ 病院で$]\ [_{VP}$ もらった$]]]\ [_{NP}$ 薬$]]\ [_P$ を$]]\ [_{VP}$ 飲んだ$]]$

```
# [$_{S}$ [$_{PP}$ [$_{NP}$ [$_{VP}$ [$_{PP}$ ヒロシが]
#                                 [$_{VP}$ [$_{PP}$ 病院で]
#                                          [$_{VP}$ もらった]]]
#                          [$_{NP}$ 薬]]
#                 [$_{P}$ を]]
#         [$_{VP}$ 飲んだ]]
```

2　下準備

以下のように L と R を統合し LHS を求め解析を進める．

```
n <- 5
example <- array(dim = c(n, n, 1))
for(d in 1:(n-1)){
    for (i in 1:(n-d)){
        j <- i + d
        for (k in i:(j-1)){
            Sys.sleep(1)
            example[ i ,j,1] <- "LHS"
            example[ i ,k,1] <- "L"
            example[k+1,j,1] <- "R"
            print(example)
            print(sprintf("d is %d, i is %d, j is %d, and k is %d",
                          d, i, j, k))
            example[ i ,j, 1] <- NA
            example[ i ,k, 1] <- NA
            example[k+1,j, 1] <- NA
        }
    }
}
```

CYK 法で肝心なのは規則 (“S -> PP VP”) を適用する，つまり右辺 (RHS) の左右を左辺 (LHS) に統合する手順である． まず $a_{i,j}$ を LHS として解析する場合を考える．その場合，$a_{i,i}$ から $a_{j,j}$ を包括する RHS が必要となる．そのような RHS は $a_{i,i}$ から $a_{j,j}$ まで動く点 k を仮定すると $a_{i,k}$ が RHS の左となり，$a_{k+1,j}$ が RHS のの右側となることが分かる．

つまり，最初の for には解析する層が該当し，これには `1:(n-1)` が該当し `d` とする．右に進んだ分 (`d`) だけ，走査する層の範囲 $a_{i,j}$ は大きくなり (`j <- i + d`)，動ける i の範囲はどんどんと狭くなる (`i in 1:(n-d)`，つまり `d` が進むほど i は小さくなる．)．そして `i` から `j-1` の間の `k` を回すことで層の for(d)，三角の移動の for(i,j), そして三角内の分割の for(k) を回せることになる．

L と R を統合するルールを適用すればいいが， 下準備がいくらか必要となる．

2.1 文法

CYK 法のような上昇型文法解析を実行するためには 1. 左辺と右辺から成る文法と， 2. 右辺から左辺を取得する関数 の 2 つがまず必要となる．まず，この節では与えられた文法を 1. 句構造規則 `rule.p` と 2. 辞書規則 `rule.d` に分け， Char 型で定義する．非終端記号を左辺に持ち，右辺には 2 つの非終端記号を持つ文脈自由文法，チョムスキー標準形を用いる．なお，辞書規則は前終端記号 (品詞) から 終端記号への遷移を示す．

```
# 句構造 (phrase) 規則
rule.p <- c("S -> PP VP" ,  "PP -> NP P",
            "NP -> VP NP",  "VP -> PP VP")
# 辞書 (dictionary) 規則
rule.d <- c("NP -> 薬",     "NP -> 病院",     "NP -> ヒロシ",
            "VP -> 飲んだ", "VP -> もらった",
            "P -> が",      "P -> で",        "P -> を")
```

次に，上の各規則に含まれる `->`で左辺と右辺を分割し Char 型の vector を生成する関数 `char2cfg` も定義する．この `char2cfg` 関数に “S -> PP VP” という文字列を与えた場合， 左辺の “S” が第一要素に，右辺の “PP VP” が第二要素に返される．

```
char2cfg <- function(s) unlist(strsplit(s, " -> "))
char2cfg("S -> PP VP")
# [1] "S"     "PP VP"
```

上で行なった規則の定義には `c` 関数を使っているが，`c` 関数の “c” は “concatenate” を意味し，ベクトルの各要素が同型であることを保証できる． 先ほど定義した `char2cfg` は Char 型をベクトルに変換する関数だった． ここで `sapply` 関数を使えば Char 型のベクトルである `rule.p` や `rule.d` の各要素に対し， `char2cfg` を apply できる．したがって，`rule.p` の各要素に `char2cfg` を `sapply` すると以下のような結果となる．

```
rule.p
# [1] "S -> PP VP"  "PP -> NP P"  "NP -> VP NP" "VP -> PP VP"
sapply(rule.p, char2cfg)
#      S -> PP VP PP -> NP P NP -> VP NP VP -> PP VP
# [1,] "S"        "PP"       "NP"        "VP"
# [2,] "PP VP"    "NP P"     "VP NP"     "PP VP"
```

2.2 統合操作

CYK 解析は上昇型の解析であるため，‘太郎’ を見たら ‘NP’ を返し，’PP’ と’VP’ を見れば統合し’S’ を返してくれるような，「右辺を統合し左辺を求める機能」が必要となる．先ほどの char2cfg を規則に sapply すると，各リストの要素には LHS と RHS が格納された．このリストを vector.p に格納し，後でこれらの行に LHS と RHS の名前をつける．

```
vector.p <- sapply(rule.p, char2cfg)
vector.p
#      S -> PP VP PP -> NP P NP -> VP NP VP -> PP VP
# [1,] "S"        "PP"       "NP"        "VP"
# [2,] "PP VP"    "NP P"     "VP NP"     "PP VP"
```

そして t 関数で転置し as.data.frame で data.frame に変換する．この「規則のベクトルから規則のテーブルへの変換 (vector2table)」を句構造規則 rule.p 対して行なうと table.p という data.frame が作れる．

```
t(vector.p)
#             [,1] [,2]
# S -> PP VP  "S"  "PP VP"
# PP -> NP P  "PP" "NP P"
# NP -> VP NP "NP" "VP NP"
# VP -> PP VP "VP" "PP VP"
vector2table <- function(l) {
    dimnames(l) <- list(c("LHS","RHS"), NULL)
    as.data.frame(t(l), stringsAsFactors=FALSE) }
table.p <- vector2table(vector.p)
table.p
#   LHS   RHS
# 1   S PP VP
# 2  PP  NP P
# 3  NP VP NP
# 4  VP PP VP
```

上の様なテーブルがあれば，RHS を参照，統合し LHS を取得することが容易となる．例えば，仮に “PP VP” を統合する規則が欲しい場合を考える．その際は「RHS 列が “PP VP” であるデータフレーム」を subset として取り，そのデータフレームの LHS 行を取得すれば良い．そのような関数を以下に rhs2lhs と定義する．

```
subset(table.p, table.p$RHS=="PP VP")
#             LHS   RHS
# S -> PP VP    S PP VP
# VP -> PP VP  VP PP VP
rhs2lhs <- function(table) function(rhs){
    subset(table, table$RHS==rhs)$LHS}
```

この関数は引数をとって関数を返す， いわゆる高階関数である．この関数はまず統合の可否を決めるために

参照するテーブルを table として引数に取り，以下の性質を持つ関数を返す．

RHS を Char 型として引数に取り，先に部分適用している table の LHS を返す．

仮に table.p を与えてから “PP VP” を与えた場合，“S” と “VP” の 2 つがベクトルとして返される． これは “PP VP” が可能な統合先には “S” と “VP” があることを示している．

```
table.p
#   LHS   RHS
# 1   S PP VP <- これと
# 2  PP  NP P
# 3  NP VP NP
# 4  VP PP VP <- これが subset となる
rhs2lhs(table.p)("PP VP")
# [1] "S"  "VP"
```

以上の操作を辞書規則でも行ない，vector.d と table.d を作成する．これらにより，単語から前終端記号へのマップが可能になる．

```
vector.d <- sapply(rule.d, char2cfg)
table.d <- vector2table(vector.d)
table.d
#   LHS      RHS
# 1  NP       薬
# 2  NP     病院
# 3  NP   ヒロシ
# 4  VP   飲んだ
# 5  VP もらった
# 6   P       が
# 7   P       で
# 8   P       を
```

まず，半角スペースので与えられた Char 型を vector に変換する関数 char2vector を考えた時， 問題となっている “ヒロシ が 病院 でもらった 薬 を 飲んだ” は Char 型のベクトル vector.input に変換できる． そして「先ほど作った rhs2lhs に辞書 (table.d) を部分適用して作った関数」にこのベクトル vector.input を sapply すると，CYK に必要な対角ベクトルが得られる．

```
char2vector <- function(s) unlist(strsplit(s, "[ ]"))
input <- "ヒロシ が 病院 で もらった 薬 を 飲んだ"
vector.input <- char2vector(input)
vector.input
# ヒロシ   が   病院   で   もらった   薬   を   飲んだ

class(rhs2lhs(table.d))
# [1] "function"
sapply(vector.input, rhs2lhs(table.d))
# ヒロシ   が   病院   で   もらった   薬   を   飲んだ
```

```
#   "NP"  "P"   "NP"   "P"      "VP"  "NP"  "P"    "VP"
```

2.3 三角行列

以上で規則に関する道具は揃ったため，CYK で使う行列に移る．CKY 法は三角行列の $a_{i,j}$ をベースに構文解析を進めていくため，最後の準備として三角行列 (対角線で区切られている行列) が必要となる．行列の i も j も長さは入力文字列がもつ前終端記号数 (今回の場合は 8) となる． なお，行列内に行列を含める言語もあるが，今回は配列，つまり行列を多次元に拡張したデータ構造の使用が好ましい．そこで array 関数を使った 8x8x1 の配列を triangle として作る．イメージとしては，8x8 の行列が 1 枚ある感じである．

```
char2vector <- function(s) unlist(strsplit(s, "[ ]"))
input <- "ヒロシ が 病院 で もらった 薬 を 飲んだ"
vector.input <- char2vector(input)
vector.input
# [1] "ヒロシ" "が" "病院"   "で"  "もらった" "薬"    "を"
# [8] "飲んだ"
n <- length(vector.input)
# [1] 8
triangle <- array(dim = c(n, n, 1))
triangle
# , , 1
#
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
# [1,]   NA   NA   NA   NA   NA   NA   NA   NA
# [2,]   NA   NA   NA   NA   NA   NA   NA   NA
# [3,]   NA   NA   NA   NA   NA   NA   NA   NA
# [4,]   NA   NA   NA   NA   NA   NA   NA   NA
# [5,]   NA   NA   NA   NA   NA   NA   NA   NA
# [6,]   NA   NA   NA   NA   NA   NA   NA   NA
# [7,]   NA   NA   NA   NA   NA   NA   NA   NA
# [8,]   NA   NA   NA   NA   NA   NA   NA   NA
```

まずは rhs2lhs を終端記号 (単語) に sapply して前終端記号のベクトル，pos とする．これを上の行列の i 行目 j 列目の triangle に組み込む操作を一般化すると，文字列の i 番目の要素を行列の i 行目，i 列目に挿入 (<<-)，という操作になる．

```
pos <- sapply(vector.input, rhs2lhs(table.d))
pos
#   ヒロシ   が    病院    で もらった     薬    を    飲んだ
#     "NP"  "P"   "NP"  "P"     "VP"  "NP"  "P"     "VP"
mapply((function(x,i) triangle[i,i,1] <<- x),
       pos, 1:(length(pos)))
       # global 変数への挿入をするため仕方なく， <<- を使っている．
triangle
# , , 1
```

```
#
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
# [1,] "NP" NA   NA   NA   NA   NA   NA   NA
# [2,] NA   "P"  NA   NA   NA   NA   NA   NA
# [3,] NA   NA   "NP" NA   NA   NA   NA   NA
# [4,] NA   NA   NA   "P"  NA   NA   NA   NA
# [5,] NA   NA   NA   NA   "VP" NA   NA   NA
# [6,] NA   NA   NA   NA   NA   "NP" NA   NA
# [7,] NA   NA   NA   NA   NA   NA   "P"  NA
# [8,] NA   NA   NA   NA   NA   NA   NA   "VP"

triangle[1,1,]
# [1] "NP"
```

2.4 ライブラリーのインポート

最後に配列を扱う上で便利なライブラリーを一つ導入する．外部パッケージには `rbind` を配列にも拡張した `abind` を利用する．

```
install.packages("abind")
library(abind)
```

3 CYK

概要にて CYK のアルゴリズムをおおかた確認したが，実際に動かす段階では幾つか問題がある． 例えば a_{ik} や a_{k+1j} の中身が一つとは限らない点や確率の情報を組み込めるかなどの拡張性の有無がある．そこで今回は行列ではなく配列を使用し， 各 a の座標に奥行きを持たせた．したがって，a_{ik} にあるベクトルが欲しければ `[i,k,]` と指定することでベクトルを取得している．そうして右辺の左のベクトル `ik`，右のベクトル `k1.j` をとり，`product <- c(outer(ik, k1.j, FUN=paste))` として組み合わせを作って `rhs2lhs` を適用している[*1]．結果はベクトルだが，配列の奥行きとしてベクトルを保存した．

```
n <- length(vector.input)
triangle <- array(dim = c(n, n, 1))
mapply((function(x,i) triangle[i,i,1] <<- x),
       pos, 1:(length(pos)))
triangle


# CYK の手順の確認

example <- array(dim = c(n, n, 1))
```

[*1] https://stackoverflow.com/questions/29959759/r-language-cross-product-combination-of-two-string-arrays や https://www.rdocumentation.org/packages/base/versions/3.5.1/topics/outer を参考にした．

```
for(d in 1:(n-1)){
    for (i in 1:(n-d)){
        j <- i + d
        for (k in i:(j-1)){
            Sys.sleep(1)
            example[ i ,j,1] <- "LHS"
            example[ i ,k,1] <- "L"
            example[k+1,j,1] <- "R"
            print(example)
            print(sprintf("d is %d, i is %d, j is %d, and k is %d",
                          d, i, j, k))
            example[ i ,j, 1] <- NA
            example[ i ,k, 1] <- NA
            example[k+1,j, 1] <- NA
        }
    }
}
# CYK の実行
for(d in 1:(n-1)){
    for (i in 1:(n-d)){
        j <- i + d
        for (k in i:(j-1)){
            ik   <- triangle[ i , k, ]
            k1.j <- triangle[k+1, j, ]
            product <- c(outer(ik, k1.j, FUN=paste))
            m <- unlist(sapply(product, rhs2lhs(table.p)))
            if(length(m)==1){
                triangle[i,j,1] <- m
            }else if(length(m)>1){
                len.m <- length(m)
                tmp <- array(dim = c(n, n, len.m))
                tmp[i,j,1:len.m] <- m
                triangle <- abind(triangle,tmp)
            }
        }
    }
}
# 最後の出力は配列の奥行きの要素を文字列として結合するのみである.
cat.table <- array(dim = c(n, n))
for(i in 1:8){
    for(j in 1:8){
    cat.table[i,j]<- toString(na.omit(triangle[i,j,]))
    }
}
```

```
cat.table
#      [,1] [,2] [,3] [,4] [,5]    [,6] [,7] [,8]
# [1,] "NP" "PP" ""   ""   "S, VP" "NP" "PP" "S, VP, S, VP, S, VP"
# [2,] ""   "P"  ""   ""   ""      ""   ""   ""
# [3,] ""   ""   "NP" "PP" "S, VP" "NP" "PP" "S, VP, S, VP"
# [4,] ""   ""   ""   "P"  ""      ""   ""   ""
# [5,] ""   ""   ""   ""   "VP"    "NP" "PP" "S, VP"
# [6,] ""   ""   ""   ""   ""      "NP" "PP" "S, VP"
# [7,] ""   ""   ""   ""   ""      ""   "P"  ""
# [8,] ""   ""   ""   ""   ""      ""   ""   "VP"
```