KoRpus

Yukio Tono

11/23/2020

0. Introduction

This brief introduction to the `KoRpus` package is based on the tutorial "Using the koRpus Package for Text Analysis" by Meik Michalke, accessed at the following URL:

https://cran.r-project.org/web/packages/koRpus/vignettes/koRpus_vignette.html#tokenizing-and-pos-tagging

Below, I applied the package to the analysis of sample texts from the NICE3.3 data to supplement my class tutorial on lexical diversity measures.

1. Basic installation

```
options(repos="https://cran.ism.ac.jp/")
install.packages("koRpus")
```

```
##
## The downloaded binary packages are in
##  /var/folders/k8/p5vmtvg11r3bhykkx0gr64380000gn/T//RtmpxzwMTf/downloaded_packages
```

```
library(koRpus)
```

```
## Warning: package 'koRpus' was built under R version 4.0.2
```

```
## Loading required package: sylly
```

```
## Warning: package 'sylly' was built under R version 4.0.2
```

```
## For information on available language packages for 'koRpus', run
##
##   available.koRpus.lang()
##
## and see ?install.koRpus.lang()
```

```
install.koRpus.lang(lang=c("en"))
```

```
##
## The downloaded binary packages are in
##  /var/folders/k8/p5vmtvg11r3bhykkx0gr64380000gn/T//RtmpxzwMTf/downloaded_packages
```

```
library(koRpus.lang.en)
```

```
## Warning: package 'koRpus.lang.en' was built under R version 4.0.2
```

2. TreeTagger installation

TreeTagger has to be installed separately.

See here: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

1. Download the tagger package for your system (PC-Linux, Mac OS-X, ARM64, ARMHF, ARM-Android, PPC64le-Linux).
2. If you have problems with your Linux kernel version, download this older Linux version and rename it to tree-tagger-linux-3.2.2.tar.gz.
3. Download the tagging scripts into the same directory.
4. Download the installation script install-tagger.sh.
5. Download the parameter files for the languages you want to process.
6. Open a terminal window and run the installation script in the directory where you have downloaded the files:
   sh install-tagger.sh
   Make a test, e.g.
   echo 'Hello world!' | cmd/tree-tagger-english
   or
   echo 'Das ist ein Test.' | cmd/tagger-chunker-german

3. Preparing Texts

```
tagged.text <- treetag(
  "koRpus/JPN501.txt",
  treetagger="manual",
  lang="en",
  TT.options=list(
    path="/Users/tono/Desktop/TreeTagger",
    preset="en"
  ),
  doc_id="sample"
)
tagged.text
```

```
##     doc_id       token  tag       lemma lttr      wclass desc stop stem idx
## 1   sample        What  DTQ        what    4  determiner <NA> <NA> <NA>   1
## 2   sample        kind  NN1        kind    4        noun <NA> <NA> <NA>   2
## 3   sample          of  PRF          of    2 preposition <NA> <NA> <NA>   3
## 4   sample      sports  NN2       sport    6        noun <NA> <NA> <NA>   4
## 5   sample          do  VBB          do    2        verb <NA> <NA> <NA>   5
```

```
## 6   sample         you  PNP         you    3     pronoun <NA> <NA> <NA>   6
##                                                    [...]
## 386 sample traditional  AJ0 traditional   11   adjective <NA> <NA> <NA> 386
## 387 sample      sports  NN2       sport    6        noun <NA> <NA> <NA> 387
## 388 sample           "  PUQ           "    1 punctuation <NA> <NA> <NA> 388
## 389 sample        BUDO  NN1        budo    4        noun <NA> <NA> <NA> 389
## 390 sample           "  PUQ           "    1 punctuation <NA> <NA> <NA> 390
## 391 sample           . SENT           .    1    fullstop <NA> <NA> <NA> 391
##     sntc
## 1      1
## 2      1
## 3      1
## 4      1
## 5      1
## 6      1
##
## 386   30
## 387   30
## 388   30
## 389   30
## 390   30
## 391   30
```

3. Descriptive statistics

All results of both `treetag()` and `tokenize()` also provide various descriptive statistics calculated from the analyzed text. You can get them by calling `describe()` on the object:

```
describe(tagged.text)
```

```
## $all.chars
## [1] 1767
##
## $lines
## [1] 30
##
## $normalized.space
## [1] 1763
##
## $chars.no.space
## [1] 1445
##
## $punct
## [1] 72
##
## $digits
## [1] 0
##
## $letters
##  all   l1   l2   l3   l4   l5   l6   l7   l8   l9  l10  l11
## 1373    2   32   80   90   51   24   24   11    1    1    3
##
```

```
## $letters.only
## [1] 1373
##
## $char.distrib
##                1          2         3         4         5          6
## num      74.00000  32.000000  80.00000  90.00000  51.00000  24.000000
## cum.sum  74.00000 106.000000 186.00000 276.00000 327.00000 351.000000
## cum.inv 317.00000 285.000000 205.00000 115.00000  64.00000  40.000000
## pct      18.92583   8.184143  20.46036  23.01790  13.04348   6.138107
## cum.pct  18.92583  27.109974  47.57033  70.58824  83.63171  89.769821
## pct.inv  81.07417  72.890026  52.42967  29.41176  16.36829  10.230179
##                7          8          9          10          11
## num      24.000000  11.000000   1.0000000   1.0000000   3.0000000
## cum.sum 375.000000 386.000000 387.0000000 388.0000000 391.0000000
## cum.inv  16.000000   5.000000   4.0000000   3.0000000   0.0000000
## pct       6.138107   2.813299   0.2557545   0.2557545   0.7672634
## cum.pct  95.907928  98.721228  98.9769821  99.2327366 100.0000000
## pct.inv   4.092072   1.278772   1.0230179   0.7672634   0.0000000
##
## $lttr.distrib
##                  1         2         3         4         5          6
## num       2.0000000  32.00000  80.00000  90.00000  51.00000  24.000000
## cum.sum   2.0000000  34.00000 114.00000 204.00000 255.00000 279.000000
## cum.inv 317.0000000 285.00000 205.00000 115.00000  64.00000  40.000000
## pct       0.6269592  10.03135  25.07837  28.21317  15.98746   7.523511
## cum.pct   0.6269592  10.65831  35.73668  63.94984  79.93730  87.460815
## pct.inv  99.3730408  89.34169  64.26332  36.05016  20.06270  12.539185
##                7          8          9          10          11
## num      24.000000  11.000000   1.0000000   1.0000000   3.0000000
## cum.sum 303.000000 314.000000 315.0000000 316.0000000 319.0000000
## cum.inv  16.000000   5.000000   4.0000000   3.0000000   0.0000000
## pct       7.523511   3.448276   0.3134796   0.3134796   0.9404389
## cum.pct  94.984326  98.432602  98.7460815  99.0595611 100.0000000
## pct.inv   5.015674   1.567398   1.2539185   0.9404389   0.0000000
##
## $words
## [1] 319
##
## $sentences
## [1] 30
##
## $avg.sentc.length
## [1] 10.63333
##
## $avg.word.length
## [1] 4.304075
##
## $doc_id
## [1] "sample"
```

Amongst others, you will find several indices describing the number of characters:

- `all.chars`: Counts each character, including all space characters

- `normalized.space`: Like all.chars, but clusters of space characters (incl. line breaks) are counted only as one character
- `chars.no.space`: Counts all characters except any space characters
- `letters.only`: Counts only letters, excluding(!) digits (which are counted seperately as digits)

You'll also find the number of `words` and `sentences`, as well as average word and sentence lengths, and tables describing how the word length is distributed throughout the text (`lttr.distrib`).

4. Lexical Diversity

To analyze the lexical diversity of our text we can now simply hand over the tagged text object to the `lex.div()` method. You can call it on the object with no further arguments (like `lex.div(tagged.text)`), but in this example we'll limit the analysis to a few measures:

```
lex.div(
  tagged.text,
  measure=c("TTR", "MSTTR", "MATTR","HD-D", "MTLD", "MTLD-MA"),
  char=c("TTR", "MATTR","HD-D", "MTLD", "MTLD-MA")
)
```

```
## Language: "en"

## MTLDMA: Calculate MTLD-MA values

##   |                                                                      |
```

```
##
## TTR characteristics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4267  0.4396  0.4938  0.5563  0.6667  1.0000
##    SD
##  0.1387
##
##
## Mean Segmental Type-Token Ratio
##              MSTTR: 0.61
##         SD of TTRs: 0.08
##       Segment size: 100
##     Tokens dropped: 19
##
## Hint: A segment size of 106 would reduce the drop rate to 1.
##       Maybe try ?segment.optimizer()
##
##
## Moving-Average Type-Token Ratio
##              MATTR: 0.57
##         SD of TTRs: 0.07
##        Window size: 100
##
## MATTR characteristics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.5292  0.5357  0.5527  0.5617  0.5722  0.6340
##    SD
##  0.0316
##
##
## HD-D
##               HD-D: 33.34
##               ATTR: 0.79
##        Sample size: 42
##
## HD-D characteristics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   31.59   32.37   30.61   32.91   33.85
##    SD
##  5.8493
##
##
## Measure of Textual Lexical Diversity
##               MTLD: 38.27
##  Number of factors: 8.33
##        Factor size: 0.72
##   SD tokens/factor: 18.26 (all factors)
##                     17.09 (complete factors only)
##
## MTLD characteristics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   10.00   33.77   36.60   35.71   38.30   48.43       1
##    SD
##  6.2353
```

```
##
##
## Moving-Average Measure of Textual Lexical Diversity
##            MTLD-MA: 41.84
##   SD tokens/factor: 17.62
##          Step size: 1
##        Factor size: 0.72
##        Min. tokens: 9
##
## MTLD-MA characteristics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   10.00   34.30   36.51   32.61   38.73   50.94       1
##    SD
##  11.6674

##
## Note: Analysis was conducted case insensitive.
```

Let's look at some particular parts: At first we are informed of the total number of types, tokens and lemmas (if available). After that the actual results are being printed, using the package's `show()` method for this particular kind of object. As you can see, it prints the actual value of each measure before a summary of the characteristics.6

Some measures return more information than just their actual index value. For instance, when the Mean Segmental Type-Token Ratio is calculated, you'll be informed how much of your text was dropped and hence not examined. A small feature tool of `koRpus`, `segment.optimizer()`, automatically recommends you with a different segment size if this could decrease the number of lost tokens.

All wrapper functions have characteristics turned off by default. The following example demonstrates how to calculate and plot the classic type-token ratio with characteristics. The resulting plot shows the typical degredation of TTR values with increasing text length:

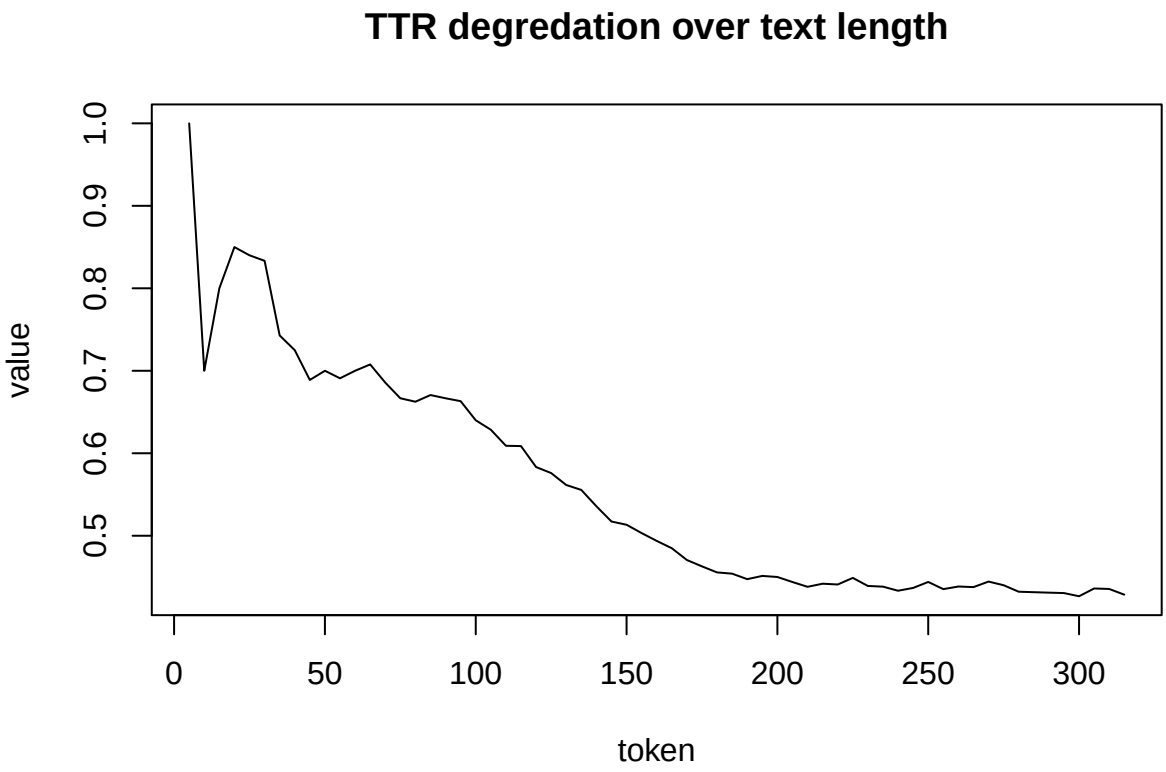
```
ttr.res <- TTR(tagged.text, char=TRUE)
```

```
## Language: "en"

## TTR.char: Calculate TTR values

##   |                                                                      |
```

```
plot(ttr.res@TTR.char, type="l", main="TTR degredation over text length")
```



5. Readability

The package comes with implementations of several readability formulae. Some of them depend on the number of syllables in the text. To achieve this, the method `hyphen()` takes objects of class `kRp.text` and applies an hyphenation algorithm (Liang, 1983) to each word. This algorithm was originally developed for automatic word hyphenation in LATEX, and is gracefully misused here to fulfill a slightly different service.

```
(hyph.txt.en <- hyphen(tagged.text))
```

```
## Hyphenation (language: en)
```

```
##   |                                                                      |
```

This seperate hyphenation step can actually be skipped, as `readability()` will do it automatically if needed. But similar to TreeTagger, `hyphen()` will most likely not produce perfect results. As a rule of thumb, if in doubt it seems to behave rather conservative, that is, is underestimates the real number of syllables in a text. This, however, would of course affect the results of several readability formulae.

So, the more accurate the end results should be, the less you should rely on the automatic hyphenation alone. But it sure is a good starting point, for there is a method called `correct.hyph()` to help you clean these results of errors later on. The most straight forward way to do this is to call `hyphenText(hyph.txt.en)`, which will get you a data frame with two colums, `word` (the hyphenated words) and `syll` (the number of syllables), in a spread sheet editor:14

```
tail(hyphenText(hyph.txt.en))
```

```
##     syll           word
## 314    1          world
## 315    1          about
## 316    1           this
## 317    4 tra-di-tion-al
## 318    1         sports
## 319    2          BU-DO
```

The hyphenated text object can now be given to `readability()`, to calculate the measures of interest:

```
readbl.txt <- readability(tagged.text, hyphen=hyph.txt.en)
```

```
## Warning: Bormuth: Missing word list, hence not calculated.
```

```
## Warning: Dale-Chall: Missing word list, hence not calculated.
```

```
## Warning: DRP: Missing Bormuth Mean Cloze, hence not calculated.
```

```
## Warning: Harris.Jacobson: Missing word list, hence not calculated.
```

```
## Warning: Spache: Missing word list, hence not calculated.
```

```
## Warning: Note: The implementations of these formulas are still subject to validation:
##   Coleman, Danielson.Bryan, Dickes.Steiwer, ELF, Fucks, Harris.Jacobson, nWS, Strain, Traenkle.Baile
##   Use the results with caution, even if they seem plausible!
##   See readability(index="validation") for more details.
```

```
readbl.txt
```

```
##
## Automated Readability Index (ARI)
##   Parameters: default
##        Grade: 4.16
##
##
## Coleman Formulas
##   Parameters: default
##     Pronouns: 10.34 (per 100 words)
```

```
##      Prepos.: 6.9 (per 100 words)
##    Formula 1: 59% cloze completions
##    Formula 2: 64% cloze completions
##    Formula 3: 66% cloze completions
##    Formula 4: 66% cloze completions
##
##
## Coleman-Liau
##   Parameters: default
##          ECP: 60% (estimted cloze percentage)
##        Grade: 6.72
##        Grade: 6.72 (short formula)
##
##
## Danielson-Bryan
##   Parameters: default
##          DB1: 6.3
##          DB2: 62.04
##        Grade: 6
##
##
## Dickes-Steiwer's Handformel
##   Parameters: default
##          TTR: 0.42
##        Score: 62.12
##
##
## Easy Listening Formula
##   Parameters: default
##       Exsyls: 77
##        Score: 2.57
##
##
## Farr-Jenkins-Paterson
##   Parameters: default
##           RE: 78.99
##        Grade: 7
##
##
## Flesch Reading Ease
##   Parameters: en (Flesch)
##           RE: 85.19
##        Grade: 6
##
##
## Flesch-Kincaid Grade Level
##   Parameters: default
##        Grade: 4.02
##          Age: 9.02
##
##
## Gunning Frequency of Gobbledygook (FOG)
##   Parameters: default
##        Grade: 6.51
```

```
##
##
## FORCAST
##   Parameters: default
##        Grade: 8.62
##          Age: 13.62
##
##
## Fucks' Stilcharakteristik
##        Score: 45.77
##        Grade: 6.77
##
##
## Linsear Write
##   Parameters: default
##   Easy words: 94.36
##   Hard words: 5.64
##        Grade: 4.92
##
##
## Läsbarhetsindex (LIX)
##   Parameters: default
##        Index: 23.17
##       Rating: very easy
##        Grade: < 5
##
##
## Neue Wiener Sachtextformeln
##   Parameters: default
##        nWS 1: 1.14
##        nWS 2: 1.86
##        nWS 3: 2.58
##        nWS 4: 2.68
##
##
## Readability Index (RIX)
##   Parameters: default
##        Index: 1.33
##        Grade: 5
##
##
## Simple Measure of Gobbledygook (SMOG)
##   Parameters: default
##        Grade: 7.55
##          Age: 12.55
##
##
## Strain Index
##   Parameters: default
##        Index: 4.18
##
##
## Tränkle-Bailer Formulas
##    Parameters: default
```

```
##  Prepositions: 7%
##  Conjunctions: 8%
##          TB 1: 52.53
##          TB 2: 51.84
##
##
## Kuntzsch's Text-Redundanz-Index
##   Parameters: default
##  Short words: 242
##  Punctuation: 72
##      Foreign: 0
##        Score: -83.38
##
##
## Tuldava's Text Difficulty Formula
##   Parameters: default
##        Index: 3.1
##
##
## Wheeler-Smith
##   Parameters: default
##        Score: 25.67
##        Grade: 3
##
## Text language: en
```

To get a more condensed overview of the results try the `summary()` method:

```
summary(readbl.txt)
```

```
## Text language: en

##                    index     flavour    raw grade  age
## 1                    ARI                    4.16
## 2             Coleman C1                59
## 3             Coleman C2                64
## 4             Coleman C3                66
## 5             Coleman C4                66
## 6           Coleman-Liau                60  6.72
## 7    Danielson-Bryan DB1               6.3
## 8    Danielson-Bryan DB2             62.04     6
## 9         Dickes-Steiwer             62.12
## 10                   ELF              2.57
## 11 Farr-Jenkins-Paterson             78.99     7
## 12               Flesch en (Flesch)  85.19     6
## 13        Flesch-Kincaid                     4.02    9
## 14                   FOG                    6.51
## 15               FORCAST                    8.62 13.6
## 16                 Fucks             45.77  6.77
## 17         Linsear-Write                    4.92
## 18                   LIX             23.17   < 5
## 19                 nWS1                     1.14
## 20                 nWS2                     1.86
```

```
## 21                  nWS3                    2.58
## 22                  nWS4                    2.68
## 23                   RIX              1.33    5
## 24                  SMOG                    7.55 12.6
## 25                Strain              4.18
## 26   Traenkle-Bailer TB1             52.53
## 27   Traenkle-Bailer TB2             51.84
## 28                   TRI            -83.38
## 29               Tuldava              3.1
## 30         Wheeler-Smith             25.67    3
```

The `summary()` method supports an additional flat format, which basically turns the table into a named numeric vector, using the raw values (because all indices have raw values, but only a few more than that). This format comes very handy when you want to use the output in further calculations:

```
summary(readbl.txt, flat=TRUE)
```

```
##                   ARI            Coleman.C1            Coleman.C2
##                  4.16                 59.00                 64.00
##            Coleman.C3            Coleman.C4          Coleman.Liau
##                 66.00                 66.00                 60.00
##   Danielson.Bryan.DB1   Danielson.Bryan.DB2        Dickes.Steiwer
##                  6.30                 62.04                 62.12
##                   ELF Farr.Jenkins.Paterson                Flesch
##                  2.57                 78.99                 85.19
##        Flesch.Kincaid                   FOG               FORCAST
##                  4.02                  6.51                  8.62
##                Fucks         Linsear.Write                   LIX
##                 45.77                  4.92                 23.17
##                 nWS1                  nWS2                  nWS3
##                  1.14                  1.86                  2.58
##                 nWS4                   RIX                  SMOG
##                  2.68                  1.33                  7.55
##               Strain   Traenkle.Bailer.TB1   Traenkle.Bailer.TB2
##                  4.18                 52.53                 51.84
##                  TRI               Tuldava         Wheeler.Smith
##               -83.38                  3.10                 25.67
```

If you're interested in a particular formula, again a wrapper function might be more convenient:

```
flesch.res <- flesch(tagged.text, hyphen=hyph.txt.en)
lix.res <- LIX(tagged.text)   # LIX doesn't need syllable count
lix.res
```

```
##
## Läsbarhetsindex (LIX)
##   Parameters: default
##        Index: 23.17
##       Rating: very easy
##        Grade: < 5
##
## Text language: en
```