Linguistik Indonesia, February 2018, 1-36 *Volume ke-36, No. 1*

ISSN cetak 0215-4846; ISSN online 2580-2429

WORKING WITH A LINGUISTIC CORPUS USING R: AN INTRODUCTORY NOTE WITH INDONESIAN NEGATING CONSTRUCTION

[a, c] Gede Primahadi Wijaya Rajeg, [b] Karlina Denistia, and [c] I Made Rajeg
[a] Monash University, Australia; [b] Eberhard Karls University of Tübingen, Germany;
[c] Universitas Udayana, Indonesia
[a] primahadiwijaya@gmail.com; [b] karlinadenistia@gmail.com; [c] made_rajeg@unud.ac.id

Abstract

This paper demonstrates the use of R for a unified data science in corpus linguistics via a series of corpus-based analyses on Indonesian Negating Construction. The data is based on c17-million word-tokens of an online-news corpus, a part of the *Indonesian Leipzig Corpora*. We identified that *tidak* is the most frequent form in our corpus. Next, we found that *tak* has significantly higher type frequency for negated-predicates with [*ter*-X-*kan*] schema compared to *tidak*; this finding provides a quantitative nuance against a description in an Indonesian reference grammar, stating that (i) in present-day Indonesian *tidak* is also common to negate *ter*- related predicates, while (ii) the compulsoriness of *tak* to negate *ter*- predicates is a past usage. Lastly, we refine our second finding by applying *Distinctive Collexeme Analysis* to determine that *tak* strongly collocates with specific verbs predominantly in the [*ter*-X-*kan*] schema compared to *tidak*; this finding offers a deeper characterisation for *tidak* and *tak*.

Keywords: *R programming language*; *Quantitative Corpus Linguistics*; *Distinctive Collexeme Analysis*; *Indonesian Negating Constructions*

Abstrak

Makalah ini menampilkan penggunaan R untuk pengolahan data holistik dalam kajian Linguistik Korpus. Sebagai contoh, makalah ini melaporkan serangkaian kajian korpus terhadap Konstruksi Negasi di Bahasa Indonesia menggunakan korpus suratkabar daring yang menjadi bagian dari *Korpus Bahasa Indonesia Leipzig*. Hasil analisis menunjukkan bahwa *tidak* adalah bentuk yang paling sering digunakan. Selanjutnya, ditemukan bahwa *tak* memiliki jumlah tipe predikat berskema [*ter*-X-*kan*] yang secara signifikan lebih banyak dibandingkan *tidak*; sudut pandang kuantitatif ini menawarkan asumsi berbeda terhadap pemaparan buku tatabahasa Bahasa Indonesia bahwa (i) *tidak* dalam Bahasa Indonesia setakat ini juga lazim digunakan menegasi predikat berawalan *ter*-, sedangkan (ii) kelaziman *tak* untuk menegasi predikat berawalan *ter*- hanyalah bagian dari sistem pemakaiannya di masa lalu. Sebagai penutup, tulisan ini memperkenalkan *Analisis Koleksem Khas* untuk menemukenali bahwa *tak*, dibandingkan dengan *tidak*, kerap berkolokasi dengan verba-verba tersendiri yang kebanyakan berskema [*ter*-X-*kan*]. Pola kolokasi khas ini menawarkan tataran yang lebih mendalam guna mencirikan *tidak* dan *tak*.

Kata kunci: *bahasa pemrograman R*; *Linguistik Korpus Kuantitatif*; *Analisis Koleksem Khas*; *Konstruksi Negasi Bahasa Indonesia*

1 INTRODUCTION[1]

Recent development in quantitative corpus linguistics prospects the growing use of programming language for data science[2] in processing the linguistic corpus and performing the statistical analyses. *R* (R Core Team, 2017) is amongst the leading programming platform used for corpus linguistics.[3] This paper is an introductory note on the use of R programming language for doing corpus linguistic research with Indonesian corpora. The main motivation for this paper is to share our reflection, especially with our fellow Indonesian linguists, on some of the ideas that we have been learning in conducting corpus linguistics using R. Our aim is to showcase that R (Section 2.1), and its *Integrated Development Environment* (IDE) namely *RStudio* (Section 2.2), can offer a *unified* state of the art for (i) some of the core text-processing stages of an electronic linguistic corpus, (ii) exploration of a linguistic phenomenon using descriptive and analytical statistics, and (iii) generating reports for communicating the results of data analyses.

As a case study, we explore the usages of *Negating Construction*[4] in Indonesian (e.g., *tidak*, *bukan*, *jangan*, etc. (Section 4.1)) (cf., Sneddon, 2006, pp. 56–57; Sneddon, Adelaar, Djenar, & Ewing, 2010, p. 203). We chose Negating Construction because this is perhaps a less studied phenomenon in Indonesian linguistics, compared to Indonesian morphology or (morpho)syntax, *inter alia* (but see Kroeger, 2014, for a detailed discussion on *bukan* 'not'). In addition, just because we take a rather peripheral area of Indonesian linguistics, it does not mean that there is an intention to compromise the depth and breadth of some of the key aspects of R for performing quantitative corpus linguistics. We will also demonstrate that corpus-based analyses allow us to pinpoint distinctive usage patterns along several analytical dimensions, even between seemingly alternating negations (e.g., *tidak* vs. *tak*). The quantitative findings for a given dimension can be used (i) to support, contradict, or offer nuances to the hypotheses in previous description on the use of the negations in Indonesian, and (ii) to generate another new hypothesis to be further tested in future studies. Eventually, through this case study, we aspire to pave a research pathway towards a quantitative usage-based approach to Indonesian.

We will focus on three aspects of the use of the negations and mainly demonstrate how the necessary steps in analysing all these aspects can be done within an integrated R environment. Firstly, we determine the frequency of occurrence of each negation (Section 4.1). The results will be discussed in relation to previous treatment of the negations in corpus data of different genre (i.e., colloquial Jakarta Indonesian) (cf., Sneddon, 2006, pp. 56–57). Secondly, we provide a quantitative corpus-based analysis to test Sneddon et al's (2010, p. 203) hypotheses regarding the use of *tidak* 'no, not' and its variant *tak* 'no, not' in modern Indonesian (Section 4.2). We will determine the extent to which these two negations are associated with the constructional schemas for the negated verbal-predicates we focus on, namely [*me*-X-*kan*], [*di*-X-*kan*], and [*ter*-X-*kan*]. We measure the association through cross-tabulating the type frequencies (i.e., the number of unique predicates) of each schema immediately following *tidak* and *tak* in our corpus (Section 4.3). Finally, we demonstrate the application of *Distinctive Collexeme Analysis* (DCA) to measure a deeper level of usage potentially distinguishing *tidak* and *tak*, namely their verb-specific preferences (Section 4.4). The remainder of this introduction provides a brief overview of what a corpus is (based on Gries, 2009a, pp. 7–11) and ends with pointers to key references on R for linguistic research.

An electronic corpus as typically used in corpus linguistics broadly refers to a(n) (extensive) collection of machine-readable texts of spoken and written languages. These

languages are usually gathered from those that are produced in a natural communicative situation, which means that the writers or speakers of the languages have no awareness for their production of the languages with the sole aim of generating a corpus. Machine-readability of a corpus means that it is typically stored in a plain text format and UTF-8 encoding so that it can be processed across computer platforms. The corpus should also be representative and balanced of the language variety, genre, or register.

There are mainly two types of linguistic corpus. The first is a *raw corpus*. The "*raw*" in *raw corpus* means that the corpus consists of only plain texts without any linguistic annotations. Examples of linguistic annotations for a corpus text include (i) part-of-speech (POS) tagging on every word in the corpus, (ii) morphological parsing, (iii) syntactic parsing, among others. The following extract is an example of the *raw* corpus from an Indonesian short story text.[5]

```
## [1] "Ketika mereka keluar dari bandara, matahari sudah tenggelam di balik kaki langit
. Darojat dijemput oleh mobil perusahaan obat-obatan, dan Driani di jemput oleh mobil pe
rusahaan perhotelan. Di dalam mobil, Darojat berpikir, jangan-jangan dia tertarik pada D
riani, dan Driani juga berpikir jangan-jangan dia tertarik pada Darojat."
```

The second corpus type is an *annotated corpus*. As indicated previously, an *annotated* corpus contains additional information attached in the corpus, such as POS-tag, lemmas of the word, etc. From the raw corpus example above, we can create a POS-annotated corpus with the *Indonesian POS Tagger*[6] (Dinakaramani, Rashel, Luthfi, & Manurung, 2014), which can be run from R as well. We designed an R programming-script for running POS Tagger so that the POS-tag labels are suffixed to the words separated by underscore ("_"). The result of the POS-annotation is shown below. The element *mereka_PRP*, for instance, indicates that the POS Tagger tag the word *mereka* 'they' as *personal pronoun* (PRP).

```
## [1] "Ketika_SC mereka_PRP keluar_VB dari_IN bandara_NN ,_Z matahari_NN sudah_MD tengg
elam_JJ di_IN balik_VB kaki langit_NN ._Z Darojat_NNP dijemput_VB oleh_IN mobil_NN perus
ahaan_NN obat-obatan_NN ,_Z dan_CC Driani_NNP di_IN jemput_VB oleh_IN mobil_NN perusahaa
n_NN perhotelan_NN ._Z Di_IN dalam_NN mobil_NN ,_Z Darojat_NNP berpikir_VB ,_Z jangan-ja
ngan_RB dia_PRP tertarik_JJ,VB pada_IN Driani_NNP ,_Z dan_CC Driani_NNP juga_RB berpikir
_VB jangan-jangan_RB dia_PRP tertarik_JJ,VB pada_IN Darojat_NNP ._Z"
```

Several introductory textbooks on R for quantitative corpus linguistics have appeared in recent years (e.g., Baayen, 2008; Desagulier, 2017; Gries, 2009a, 2009b, 2013c; Levshina, 2015). As fellow learners, it is our hope that the interested readers, especially within the Indonesian linguistics circle, will dive into these references for further collaboration in conducting (corpus) linguistic research with R.

2 R AND R STUDIO IN A NUTSHELL

2.1 R environment and R functions

R is an open-source programming language for statistical computing and data visualisations. It can be downloaded for free from *CRAN*[7] (i.e., the *comprehensive R archive network*) and is available for Windows, Mac (OS X), and Linux. This paper will show that R is a powerful resource, not only for statistical analyses but also for one of the central tasks for corpus linguistics, namely text processing.

As an open-source software, R emerges into a *community of practice* where anyone can contribute to the development of R functionalities. R has its main conference, namely *useR,*[8] and a dedicated journal, *The R Journal.*[9] There is also an R-related blog called *R-bloggers*[10] that is the hub for R users to exchange and share updates, including tutorials, for doing data science

with R. Another excellent place to look for solutions to R-related issues is *Stack Overflow,*[11] among others. Thus, there are a lot of online sources through which one can learn about R.

Upon opening R, users will face a console (cf., Figure 1 below), which prints out the information about the R version installed and any other relevant information about the R statistical projects, including how to cite it. Following this introductory information is an empty console. It expects the users to type a command-input/programming code after the larger-than symbol (`>`) called *prompt line*. After typing the command, users then hit "Enter" for R to perform the specified command. The snippet in Figure 1 shows simple arithmetic operations as commands for R to execute. They are *addition* (i.e., `1 + 1`) and *multiplication* (i.e., `2 * 2`).

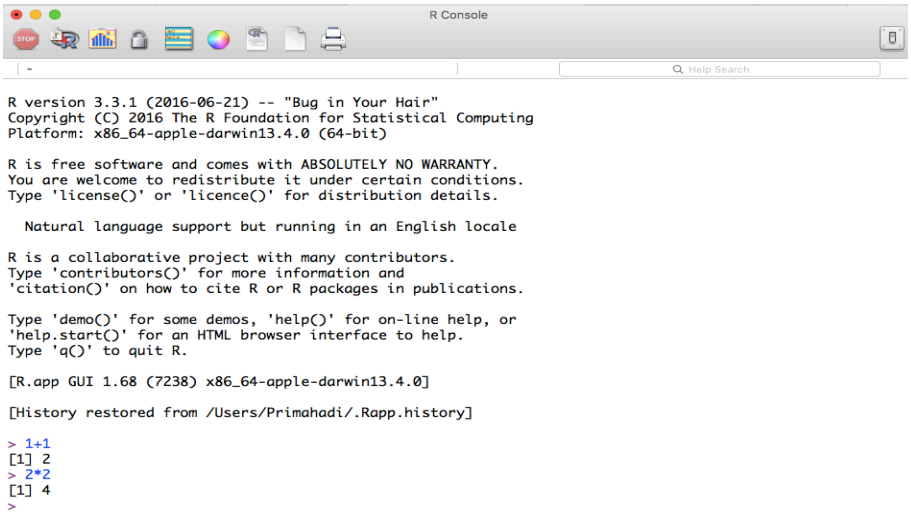

Figure 1 Snippet for R console (on Mac OS X)

Two or more commands can be executed simultaneously in one line. Each of these commands should be separated by semicolons ";". This is exemplified below.[12]

```
# the last one (i.e., "5^2") is a "raising to power" arithmetic (equal to "5*5
").
1+1; 2*2; 16/2; 5^2
## [1] 2
## [1] 4
## [1] 8
## [1] 25
```

The number in square brackets (e.g., `[1]`) indicates index number of the output element(s) of a given command. Each of the above commands produces only a single output, hence the `1` in the brackets.

R of course can do more than just simple arithmetic. In learning R as a programming *language*, one needs to know the *language* used to command R to perform certain operations. The command is essentially in the form of *functions* and their required *argument(s)*. The basic schema of an R function is `name.of.a.function(argumentA = ..., argumentB = ..., ...)` where two or more of the specified arguments are separated by commas. The argument(s) to a function indicates "(i) what the instruction to be applied to and (ii) how the instruction is to be applied to it" (Gries, 2009a, p. 24). The following example with `sqrt()` function shows an operation to derive the square root of a number (i.e., 8); `sqrt()` requires only one argument that is by default labelled as `x`.

```
# compute the square root of 8
sqrt(x = 8)

# or
sqrt(8)
## [1] 2.828427
```

Argument label(s) of a given function, such as the `x` argument for `sqrt()`, can be left out in the code provided that we remember to supply the argument(s) in the *exact order* as required by the function. For instance, input for argument `x` must come before input for argument `y`, etc. A safer practice is to spell-out the specified arguments when a function has many labelled arguments.

We can store, or assign, the output of the above command into a data structure for later use. To assign output(s) of a command, we use the arrow symbol `<-` (i.e., the combination of a less-than sign `<` and a minus `-`). To instruct R to print out or output the assigned value to the console, just type the name of the data structure we use to store the results and then hit "Enter". These operations are shown below.

```
# assign the output of `sqrt()`
ss <- sqrt(8)

# print out the result
ss
## [1] 2.828427
```

The stored output can be used for further operation. For instance, we could round the floating points of the result of the square root of eight by means of `round()`. We can use the `ss` data structure as one of the input arguments for `round()` (i.e., the `x` argument, which is the number to be rounded). Another argument of `round()` is `digits`, specifying the number of floating points to keep.[13] Then, the output of `round()` can be stored into another data structure that we may call `ss.rnd`.

```
# round the floating points
ss.rnd <- round(x = ss, digits = 2)

# output the result
ss.rnd
## [1] 2.83
```

The base installation of R has a variety of functions for a great number of data science processes, including text processing and statistical analyses. There are also functions, and data, that come with the so-called R *packages*. R packages are features that enhance the capability of the base R; all R packages are also available for free. The packages need to be installed and loaded when opening R so that we can utilise the functions from the packages. This paper will illustrate how data processing in relation to corpus linguistic analysis can be performed by functions from the basic installation of R.

2.2 Unified data science in RStudio with R Markdown

This section provides a brief tour on the integrated environment for R programming, namely *RStudio*, which is also available for free.[14] One of the hallmarks of RStudio is its *authoring* feature called *R Markdown*[15](Wickham & Grolemund, 2017, pp. 423, 469, 479), which is an integral part of the *R Notebook* interface.[16] R Markdown notebook is a platform to (i) write the programming-code for conducting our data processing and analyses, (ii) output in-line results of the programming for the analyses, and *simultaneously* (iii) write-up the narratives for the results.

R Markdown support a range of output formats, including MS Word.[17] Put it simply, we can write papers, theses, or even books, with the analysis of a phenomenon via R Markdown in RStudio; nearly every single word in this paper, the workflow for analysing the Negating Construction, and the results are all written and processed via an R Markdown notebook in RStudio. Figure 2 shows a snippet of the RStudio interface of an R Markdown notebook for the writing of this paper[18] (upper panel); it also shows the R-console window (lower panel) where the programming codes are normally executed.

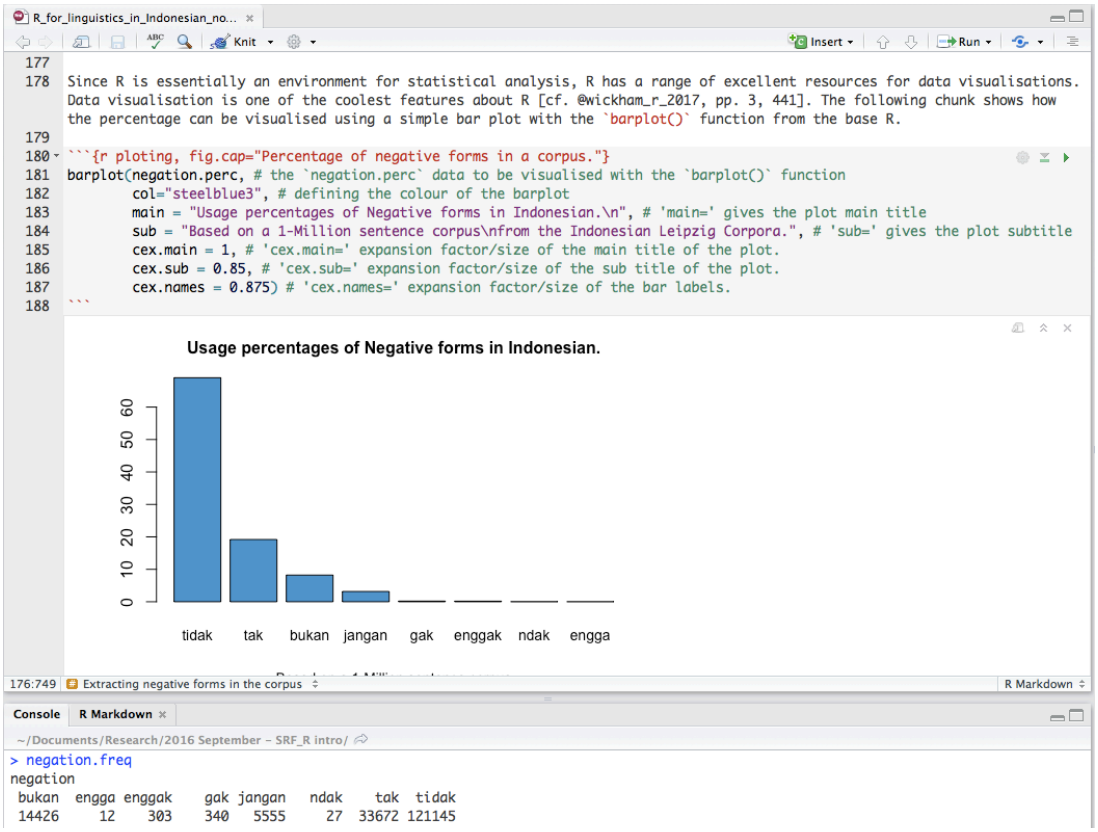

Figure 2 RStudio snippet for R Markdown interface (top window) and R console (bottom window)

The grey-shaded area (starting from line 180 to 188) in the snippet above is called the *code-chunk* where we insert the R codes to perform a given operation. We have seen this code-chunk in Section 2.1 when discussing R functions. The area below the code-chunk may contain outputs of a specific operation. In the snippet above, it is illustrated with the output of a plotting/visualisation command (cf., Figure 3 below). Other typical in-line output include (i) *tables* (e.g., Table 1 in Section 4.1 below), (ii) *numbers* as we have seen before with `round()` and `sqrt()`, or (iii) *character strings/texts* (cf., Section 3.2). The narratives of the whole document are written mainly in the white area as plain text; in Figure 2 above, it is the area above the code-chunk (i.e., starting from line 177).

The snippet of the R Markdown interface above hopefully helps readers to form an overall picture of what we mean by performing a *unified*, well-tailored data science with R in RStudio. This spans across writing the programming codes for processing and analysing the data through to writing up and reporting the results within *one* environment. Such unified environment then allows us to generate the so-called *reproducible* analyses and report (cf., Flanagan, 2017). It means that all analytical steps, such as data processing, programming codes,

and their outputs, are available in the form of an R Markdown document. In this way, people having access to the R Markdown document and the dataset can reproduce what we do with the aim of obtaining the same results. Moreover, R Markdown considerably reduces pain from having separate files for the research notes, datasets, results of quantitative analysis from a statistical software, and another software to write-up the results. Practically, RStudio and R Markdown help the "future-us" to manage and recall all this complex information. In the next section, we flesh out this unified process of data analysis in R through a corpus linguistic study.

3 EXPLORING CORPUS DATA WITH R

This section demonstrates the processing of corpus data in R for corpus linguistic exploration on the Negating Construction in Indonesian (cf. Section 4); our analytical workflows are presented in the embedded code-chunks. For this paper, we use one corpus file, that is the `ind_newscrawl_2012_1M-sentences.txt`, out of the whole collection of the *Indonesian Leipzig Corpora* (Biemann, Heyer, Quasthoff, & Richter, 2007; Goldhahn, Eckart, & Quasthoff, 2012). *Leipzig Corpora* is a collection of sentence-based corpus texts with number in each sentence; they can be downloaded for free.[19] The corpus files in the Indonesian Leipzig Corpora are *raw corpus* since there are no additional annotations, such as POS-tag. The sources of the text in Leipzig Corpora are mainly online newspapers, randomly scrapped webpages, and Wikipedia.

3.1 Loading corpus files into R

The following code-chunk shows how a corpus file is loaded into R for further processing. The function to read-in a file containing texts, such as a corpus, is `scan()`.

```
# read in a UTF-8 encoding Leipzig corpus file using `scan()`
corpus <- scan(file = "/Users/Primahadi/Desktop/ind_newscrawl_2012_1M-sentence
s.txt", what = "char", sep = "\n", encoding = "UTF-8", quiet = TRUE)
```

For our purpose, `scan()` requires at least three of its arguments to be specified. The first argument is `file`, which is specified with input of the name of the corpus file, preferably with the full directory path; argument inputs of the type text/string, such as a name of a corpus file, should be quoted with double-quote or single quote. The second argument is `what`, which indicates the type of the file to be loaded; in this case, the corpus is the type of character; hence the `"char"` value. The last argument is `sep` that specifies how the text file is separated; in this case, each sentence in the Leipzig corpus is separated by newlines, hence the `"\n"` value of `sep`. If the corpus file is in UTF-8 encoding, it is important to specify the `encoding` argument as well. Meanwhile, the `quiet = TRUE` parameter is optional to suppress printing message about the total number of lines read-in from the corpus.

We can then inspect the values of the character vector `corpus` containing the loaded corpus. One of the possible operations is to find out the number of elements, in this case sentence lines, in the corpus. This can be done with `length()` with `corpus` as its main argument.

```
length(corpus)
## [1] 1000000
```

The output of `length()` indicates that the loaded Indonesian Leipzig corpus consists of 1,000,000 million elements representing sentences, which is obvious from the name of the corpus file. The `corpus` variable represents one of the basic data structures in R, namely a *vector* (cf., Gries, 2009b, pp. 66–71). A vector consists of "one-dimensional, sequentially ordered

sequences of elements (such as numbers or character strings (such as words))" (Gries, 2009b, p. 66). In Section 2.1, we have dealt with numeric vectors in the introduction to few R functions with `sqrt()` and `round()`; we will see below other types of data structures in R, namely *list* and *data frame*/*table*. We can check whether or not `corpus` is a vector using the following code.

```
# check if 'corpus' variable is a vector
is.vector(corpus)
## [1] TRUE
```

The code below checks the type of elements contained in the `corpus` vector.

```
# check the type of an object
typeof(corpus)
## [1] "character"
```

To subset/access parts of the elements in a vector (or the other data structures), we use square brackets `[ ]`. The following code-chunks illustrate few possible scenarios for subsetting elements (i.e., sentences) in the `corpus` vector. The first one is as follows.

```
# subset the first two sentences in "corpus"
# 1st possibility
corpus[1:2]
## [1] "1 Bintang sinetron “Dia Bukan Anakku” itu merasa alergi dengan kendaraan ber-AC.
"
## [2] "2 Sehingga keluar dari P3R dan memulai usaha baru dengan modal uang tabungan."
```

The second alternative for the above code is as follows.

```
# 2nd possibility
# generate numeric sequences of 1 to 2 and assign them to a vector "index"
index <- 1:2

# use the content of vector "index" to subset the "corpus"
corpus[index]
## [1] "1 Bintang sinetron “Dia Bukan Anakku” itu merasa alergi dengan kendaraan ber-AC.
"
## [2] "2 Sehingga keluar dari P3R dan memulai usaha baru dengan modal uang tabungan."
```

Lastly, the following code shows how to subset vector elements in a mixed sequence. For instance, subsetting the second, seventh, and ninth to eleventh elements of the `corpus`. We use the `c()` function to *concatenate*/*join* several elements (e.g., the subsetting number-indices) into the `index` vector.

```
# create subsetting indexes
index <- c(2, 7, 9:11) # 1st option
index <- c(2, 7, 9, 10, 11) # 2nd option

# subset the 2nd, 7th, and 9th to 11th elements of 'corpus'
corpus[index]
## [1] "2 Sehingga keluar dari P3R dan memulai usaha baru dengan modal uang tabungan."
## [2] "7 Pengumuman MK bahwa Jaksa Agung Hendarman Supandji tidak sah, juga terkesan sa
rat muatan politis."
## [3] "9 Karena itu, warga Ngada tidak akan pernah mengalami rawan pangan ataupun busun
g lapar."
## [4] "10 Menurutnya, dimanapun Djoko Susilo ditahan, yang penting tempat itu harus ste
ril dan kondusif."
## [5] "11 Ia selalu bertanya pada bayangan bertopeng dalam cermin ajaibnya."
```

The output shows that we do get sentences number 2, 7, 9, 10, and 11; the numbers in the square brackets show that for the above command, there are five sequential outputs (hence, `[1]` to `[5]`).

3.2 Determining the size of a corpus

Before coming to our target construction, we would like to demonstrate how one can determine the total size (in a word-token) of our corpus, that is how many words are there in a one-million sentence corpus? To do that, firstly, we need to tokenise, or split, character strings, such as sentences, into individual words. The relevant function for this purpose is `strsplit()`.

```
# define the pattern to tokenise the sentences
split.regex <- "([^a-zA-Z0-9-]+|--)"

# tokenise/split the corpus; this takes a couple of seconds
wordtoken.list <- strsplit(x = corpus, split = split.regex, perl = TRUE)
```

The `x` argument of `strsplit()` requires the texts/character strings to be split; in this case, it is our corpus stored in the vector `corpus`. The `split` argument specifies a pattern for how the texts to be split; for this purpose, we use a powerful pattern-matching procedure called *regular expressions* (henceforth, *regex*) (Gries, 2009a, pp. 79–99; Sanchez, 2013; Wickham & Grolemund, 2017, pp. 200–207). Then, the `perl` argument is set to `TRUE` to indicate that we use the *Perl*[20]-compatible regex.

The regex we use to split the sentences is `"([^a-zA-Z0-9-]+|--)"` (cf., Gries, 2009a, p. 151). This regex is designed to capture and split characters that are not part of a word:

> split at any one or more (i.e., the `"+"`) characters that are (i) NOT (i.e., the caret `"^"` inside the square brackets) parts of alphanumeric character classes (i.e., `"[^a-zA-Z0-9]"`) and (ii) NOT hyphen (`"-"`, hence `"[^a-zA-Z0-9-]"`), as to maintain reduplicated tokens, such as *anak-anak* 'children', (iii) OR (i.e., the pipe `"|"`) split at characters that are dashes (`"--"`).

The `wordtoken.list` object contains the results of `strsplit()`, which are in the form of a data structure called *list*.

```
# check if it is a 'list'
is.list(wordtoken.list)
## [1] TRUE

# check the length of the 'wordtoken.list' list. It should be equal to the
# number of sentences (i.e., 1 million)
length(wordtoken.list)
## [1] 1000000
```

A *list* is a very versatile data structure since it can contain heterogeneous data structures (cf., Wickham & Grolemund, 2017, p. 302). For instance, the first element of a list can be a data frame while the other elements can be (numeric or character) vectors, or even lists. We will not deal with this in details here. However, it suffices to mention that each (one-million) element of the `wordtoken.list` contains character vectors of words derived from splitting up each (one-million) sentence. The following codes subset two specific elements within the `wordtoken.list` to show how a list looks like when it contains the split character-vector elements.

```
# subset the 481860th and 340189th elements of the list
wordtoken.list[c(481860, 340189)]
## [[1]]
## [1] "481860"   "Tidak"    "terdapat" "naskah"   "tentang"  "HB"
## [7] "VIII"
##
## [[2]]
##  [1] "340189"    "Bukan"     "sekadar"   "bahasa"    "yang"
```

```
##  [6] "bisa"      "terjebak"  "pepatah"   "lidah"     "tidak"
## [11] "bertulang"

# TRY the following codes for yourself!
# 1. return the vector of the first element of the list
wordtoken.list[[1]]

# 2. return the second element of the vector, which is in turn the first element of the list
wordtoken.list[[1]][2]
```

For the purpose of counting all the tokenised words in the corpus, it is more manageable to turn the `wordtoken.list` into an atomic vector (cf. Wickham & Grolemund, 2017, p. 293). The vector then contains elements of all words in the corpus. The relevant function for this operation is `unlist()` with a list as the input data; `unlist()` turns the complex list into an atomic vector.

```
# unlist into a long vector of words; it takes a couple of seconds
wordtoken.vect <- unlist(wordtoken.list)

# inspect the first six elements of the vector 'wordtoken.vect'
wordtoken.vect[1:6]
## [1] "1"        "Bintang"  "sinetron" "Dia"      "Bukan"    "Anakku"
```

The next step is to count the number of words in the corpus by using `length()` and feeding the `wordtoken.vect` as the input-argument.

```
# count the number of words in the corpus
length(wordtoken.vect)
## [1] 17942268
```

The value indicates that in our corpus file of one-million sentences, there are 17,942,268 million word-*tokens*; the words consist of one or more, case-insensitive, alphanumeric character, including hyphen (-), separated by whitespace. In order to get the number of unique word-*types*, we need to lowercase the words using `tolower()`, before feeding them into `unique()` then `length()`, as shown in the following codes.

```
# lower case
wordtoken.vect.lower <- tolower(wordtoken.vect)

# get unique elements/words
wordtoken.vect.lower <- unique(wordtoken.vect.lower)

# count the number of unique word-types
length(wordtoken.vect.lower)
## [1] 1261174
```

The output tells us that the 17,942,268 million word-*tokens* of the corpus are made up of 1,261,174 million word-*types*.

4 EXPLORING THE INDONESIAN NEGATING CONSTRUCTIONS

4.1 Extracting the negating forms

We now proceed to obtain the occurrence of a set of Negating Constructions in Indonesian and generate their frequency of occurrences (cf., below). The constructions include the following forms: *bukan*, *enggak*, *engga*, *gak*, *jangan*, *ndak*, *tidak*, *tak*. The forms *enggak*, *engga*, and *gak* are attested as variants at the level of pronunciation; they occur in high frequency in Colloquial Jakarta Indonesian, especially *enggak* (cf., Sneddon, 2006, pp. 56–57). Another colloquial form

is *ndak*, which is potentially a variant of the formal *tidak*. *Tak*, which is suggested to have "a literary flavour for most Indonesians", is considered another variant of *tidak* (Sneddon et al., 2010, p. 203). *Jangan* is a negation in an imperative sentence closely resembling English 'don't' while *bukan* 'not sth. (but rather sth. else)' typically negates nouns (cf., Sneddon et al., 2010, p. 334). All these forms will also be retrieved using regex that we have seen in Section 3.2 when we perform the tokenisation of the words (in the `split` argument of `strsplit()`).

We design the following regex to extract the potential match for each negative form: `"\\b(bukan|enggak|engga|gak|jangan|ndak|tidak|tak)\\b"`. The `"\\b"` part indicates the word boundaries (Gries, 2009a, p. 128); in this case, we want the form *tak*, for instance, to be extracted as such (i.e., *tak*), rather than as part of a larger word (e.g., *bo**tak*** 'bald'). The pipe `"|"` indicates alternative. The round brackets `"()"` capture the sequence of the character strings (e.g., `"bukan"`) as *a unit/group of* alphabetic characters.

The following code-chunk shows the whole process of extracting the negating forms from the corpus. The relevant functions are `gregexpr()` and `regmatches()`: the former outputs the starting and final character-position of the searched pattern, including its length (in character), meanwhile the latter extracts the relevant forms based on the output of `gregexpr()`[21]. In Section 4.2 below, we will see how the use of these two functions in the following code-chunk can be adjusted and re-cycled for a slightly different purpose, namely finding the negated predicates immediately following the negations.

```
# define the searching regex
regex <- "\\b(bukan|enggak|engga|gak|jangan|ndak|tidak|tak)\\b"

# get the starting and ending character-positions of the negative forms
posm <- gregexpr(pattern = regex, text = corpus, perl = TRUE, ignore.case = TR
UE)

# extract the exact forms using `regmatches()`
negation <- regmatches(x = corpus, m = posm)

# (Optional) remove the 'posm' list from the working space to save memory
rm(posm)

# unlist the results into an atomic character vector for easier processing
negation <- unlist(negation)

# lowercase the results
negation <- tolower(negation)

# inspect the first three matches found from the corpus
negation[1:3]
## [1] "bukan" "tidak" "tidak"
```

After retrieving the negating forms, one may proceed with generating a frequency list of all these forms. This analysis can be performed with `table()` as shown below.

```
# create frequency list of the negative forms
negation.freq <- table(negation)
```

The frequency list can then be sorted using `sort()` with its `decreasing` argument is set to `TRUE`.

```
# sort the frequency list in decreasing order
negation.freq.sorted <- sort(negation.freq, decreasing = TRUE)
```

```
# print out the results
negation.freq.sorted
## negation
##  tidak    tak  bukan jangan    gak enggak   ndak  engga
## 121145  33672  14426   5555    340    303     27     12
```

The outputs of `table()` above are raw frequency *counts/tokens* of the forms. It would also be good to provide the relative frequency of these forms in the form of proportion or percentages so that the forms can be compared along the same scale (cf., Janda & Lyashevskaya, 2013, p. 224). The following chunk shows how the raw frequencies can be converted into (i) proportions, using `prop.table()`, or (ii) percentages, by multiplying the results of `prop.table()` with 100.

```
# generate percentages of the negation
negation.perc <- prop.table(negation.freq.sorted) * 100

# rounding the percentages with '2' floating digits
negation.perc <- round(negation.perc, digits = 2)

# print out the results
negation.perc
## negation
##  tidak    tak  bukan jangan    gak enggak   ndak  engga
##  69.04  19.19   8.22   3.17   0.19   0.17   0.02   0.01
```

A basic insight that can be learnt from the relative frequencies is the extent to which a given negating form is used in a one-million sentence corpus of Indonesian online news. The colloquial forms, namely *gak*, *enggak*, *engga*, and *ndak*, occur much less frequently, which could be expected given that they have been found to be highly frequent in Colloquial Jakarta Indonesian, especially in the conversation and interview recordings (cf., Sneddon, 2006, p. 57). In terms of usage property, the usage proportion of the negation can indicate which variants are relatively more common of a given corpus genre (i.e., online news). *Tidak* is the common negative form used in the corpus, taking up 69.04% of the total occurrences of the extracted negating forms. This could be explained, in part, by the more formal setting of news genre, which would require the use of more formal variant (i.e., *tidak*).

Since R is also essentially an environment for data visualisation, R has a range of resources for visualising results of statistical analyses (cf., Wickham & Grolemund, 2017, pp. 3, 441). The following chunk shows how the percentages of the negating forms (stored in `negation.perc`) are visualised using a bar plot with `barplot()` from the base R (cf., Figure 3).

```
barplot(negation.perc, col = "steelblue3", cex.main = 1, cex.names = 0.7, ylim
= c(0, 100), main = "Usage percentages of negating forms in Indonesian.")
```

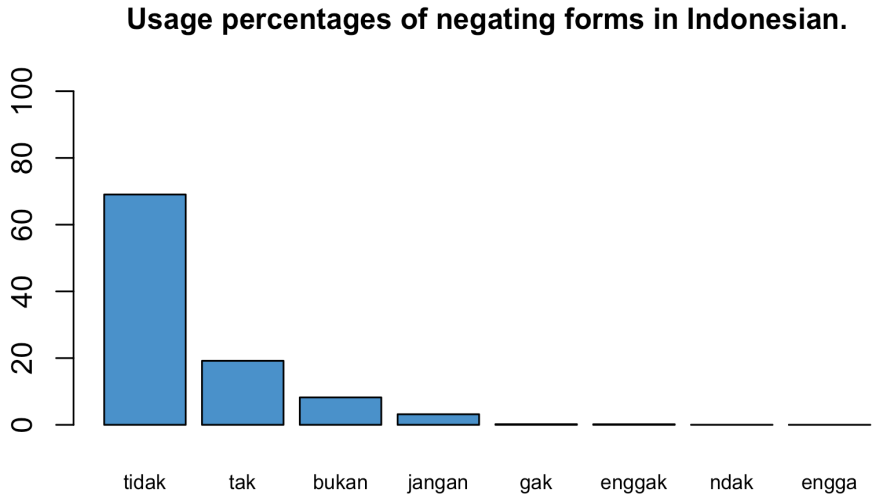

Figure 3 Percentages of negating forms in 'ind_newscrawl_2012_1M-sentences.txt' corpus of the Indonesian Leipzig Corpora

The `col` argument of `barplot()` can be filled with character name for the bar colours. The `main` argument is filled with texts for the main title of the plot. The `cex.main` and `cex.names` arguments respectively control the character expansion factor for the size of the main title and the bar labels. The `ylim` argument specifies the limit of the *y*-axis (i.e., the vertical axis) that ranges from zero to maximum 100, representing percentages of the negating forms.

Another remarkable feature of R Markdown notebook is that we can generate an in-line table as part of our report. For that purpose, we need to turn the results stored in `negation.perc` vector into a table format called *data frame* with the following codes; the relevant function to generate a data frame from a set of atomic vectors of the same length is `data.frame()`.

```
# get the percentage value as numeric vector
percentages <- as.vector(negation.perc, mode = "numeric")

# get the names of the constructions from the "negation.perc" vector
negations <- names(negation.perc)

# put the "negations" and "percentages" vectors into a data frame
negation.dframe <- data.frame(negations, percentages, stringsAsFactors = FALSE
)

# print out the data frame
negation.dframe
```

Table 1 The data frame of percentages of the Negating Constructions

	negations	percentages
1	*tidak*	69.04
2	*tak*	19.19
3	*bukan*	8.22
4	*jangan*	3.17
5	*gak*	0.19
6	*enggak*	0.17
7	*ndak*	0.02
8	*engga*	0.01

Up to this stage, we are hoping that we have been able to demonstrate a simple, yet hopefully elegant, example of how a series of data processing into gaining insight about a linguistic phenomenon can be done in a unified fashion in R. They include loading the corpus text, extract the occurrences of the linguistic phenomenon of interest (with regexes), perform one of the core corpus linguistic analyses (viz. frequency list), and finally communicate the results, via a table (e.g., Table 1) and a plot (e.g., Figure 3). The remainder of this paper provides more intriguing examples by (i) recycling the searching procedures that we have performed, with few additions, and (ii) generating further theoretical insights into the use of the negations using a set of analytical statistics.

4.2 Extracting collocational patterns of the Negating Constructions

This section takes a step further in exploring the usage of the Negating Constructions in Indonesian. We focus on the collocational patterns of the two most frequent negation forms in our corpus, namely *tidak* and *tak*. The collocational patterns are restricted to the fillers of the negated predicates immediately to the right of these negation forms. The constructional schema of this collocation can be represented as [*tidak/tak* + PREDICATES]. As an illustration, the predicate-fillers to be explored are potential *verbal* predicates in three constructional schemas, namely [*me*-X-*kan*], [*di*-X-*kan*], and [*ter*-X-*kan*]. The "Xs" may stand for the root forms of the verbs; the prefixes indicate different grammatical voices: active (*me*-), (dynamic) passive (*di*-), and (stative) passive (*ter*-) (cf., Arka, 2010, for the discussion on dynamic and stative passive in Indonesian); the *-kan* suffix is intended to ensure the predicates are (di)transitive verbs.

The searching procedures and related functions at the beginning of Section 4.1 will be re-used to help readers internalise how a series of similar methods can be slightly tweaked for different analytical purposes. The key variation here is mainly on the regex design to be used to extract the collocation. For instance, the regex that we designed to search for the immediate negated-predicates for *tidak* is `"\\b(?<=tidak\\s)((ter|di|me)[a-z]{3,}kan)\\b"`. What the regex does is as follows:

> search the occurrence of words that begin with *ter, di,* or *me* prefixes (i.e., `"(ter|di|me)"`) followed by at least three or more alphabetic characters (i.e., `"[a-z]{3, }"`), and that are ended with *kan* (i.e., `"kan"`). These words have to be immediately preceded by *tidak* followed by a white space (i.e., `"tidak\\s"`); but the *positive lookbehind* assertion (i.e., the `"(?<=)"` part) in `"(?<=tidak\\s)"`) indicates that *tidak* and the white space (`"\\s"`) will not be captured/extracted in the output, hence only the negated predicates.

The same regex design will be used to search for the negated-predicates for *tak* by replacing `"tidak"` with `"tak"` in the regex above.

One thing to keep in mind is that, based on our intuition, we assume that the three constructional schemas restriction above and the regex design will return mostly verbs, but not the other word classes, or even other verbs, that happen to contain *ter-*/*di-*/*me-* and *-kan* strings. For instance, *ter**ia**kan* 'a shout(ing)', based on *teriak* 'to shout', will be filtered out by the regex because there are only two alphabetic characters in this word following the prefix *ter-*, namely *ia*, before followed by the suffix *-kan*. Similar case holds for verbs. For example, *me**ma**kan* (from *makan* 'to eat') and *di**te**kan* 'to be pressed' (from *tekan* 'to press') will not be captured by the regex. In this case, the results of analyses in relation to the collocational data, as reported in Section 4.3 and Section 4.4, are restricted to the potential verbs with the specified three constructional schemas as captured by the regex design. Future study can generate different

regex design to capture all potential *me-*, *di-*, and *ter-* verbs, which may include the suffix *-kan* and *-i*, negated by *tidak* and *tak*. The following codes show the processes to extract the negated predicates for *tidak*. The top-10 results are presented in Table 2.

```
# 1. search the collocates, generate the frequency and percentage counts

# define the searching regex
regex <- "(?<=\\b(tidak\\s))((ter|di|me)[a-z]{3,}kan)\\b"

# search the starting and ending character-position of the matches
posm <- gregexpr(pattern = regex, text = corpus, perl = TRUE, ignore.case = TR
UE)

# extract the match
predicates <- regmatches(x = corpus, m = posm)

# remove 'posm' list to save memory
rm(posm)

# unlist into a vector of characters
predicates <- unlist(predicates)

# lower case the match
predicates <- tolower(predicates)

# create frequency list
frequency <- table(predicates)

# sort in decreasing order
frequency <- sort(frequency, decreasing = TRUE)

# create percentage
percentage <- prop.table(frequency) * 100

# round the percentages
percentage <- round(percentage, digits = 2)

# 2. generate the data frame

# get the names of the predicates from the 'frequency' data
predicates <- names(frequency)

# get the frequency value as numeric vector
frequency <- as.vector(frequency, mode = "numeric")

# get the percentage value as numeric vector
percentage <- as.vector(percentage, mode = "numeric")

# create the data frame
tidak.coll <- data.frame(predicates, frequency, percentage, stringsAsFactors =
FALSE)

# print the first top-ten rows [1:10, ] of the table
tidak.coll[1:10, ]
```

Table 2 Top-10 most frequent negated predicates with *tidak*

	predicates	gloss	frequency	percentage
1	*melakukan*	to do	669	6.11
2	*memberikan*	to give	506	4.62
3	*dilakukan*	to be done	314	2.87
4	*menggunakan*	to use	299	2.73
5	*mendapatkan*	to receive; obtain	286	2.61
6	*menimbulkan*	to cause to emerge	271	2.48
7	*ditemukan*	to be found	229	2.09
8	*diinginkan*	to be wanted	226	2.06
9	*menemukan*	to find; discover	183	1.67
10	*menyebutkan*	to mention	158	1.44

For the reason of space, the codes for *tak* are not printed out in the paper. The codes are the same as *tidak*. One only needs to change (i) `"tidak"` with `"tak"` in the regex, and (ii) the name of the final data frame from `tidak.coll` to `tak.coll`. Table 3 shows the top-ten results for *tak*.

Table 3 Top-10 most frequent negated predicates with *tak*

	predicates	gloss	frequency	percentage
1	*mengherankan*	to (be) astound(ing)	89	4.77
2	*terkalahkan*	(can) be defeated; defeatable	87	4.67
3	*terhindarkan*	(can) be avoided; avoidable	68	3.65
4	*memberikan*	to give	51	2.74
5	*diinginkan*	to be wanted	46	2.47
6	*melakukan*	to do	44	2.36
7	*terelakkan*	(can) be avoided; avoidable	44	2.36
8	*diragukan*	to be doubted	43	2.31
9	*menemukan*	to find; discover	41	2.20
10	*menyurutkan*	to lessen; abate	41	2.20

There are at least two points that can be explored from the data of the negated-predicate collocates. First, we may investigate the total number of different types of verbs to be negated with *tak* and *tidak*. The different verbal types can further be measured, for instance, at the level of the number of the verbs conforming to the three constructional schemas mentioned above. The findings may elucidate the degree of association between the two negative forms and the three constructional schemas (cf., Section 4.3). The second analytical point is that we can rank all the negated collocates according to their statistical preferences/tendencies to be negated with *tak* in comparison to *tidak*; for this purpose, we will demonstrate the application of a quantitative corpus linguistic method called *Distinctive Collexeme Analysis* (cf., Section 4.4.1). The ranking may then show us which specific verbal predicates are relatively more distinctive to be negated with a given negation (cf., Section 4.4).

4.3 Type frequencies of the negated-predicates and their schemas

If we compare the number of rows in each data set (i.e., the `tidak.coll` and `tak.coll` data frames), *tidak*, in comparison to *tak*, has nearly three times more number of unique negated predicates conforming to the [*me*-X-*kan*], [*di*-X-*kan*], and [*ter*-X-*kan*] schemas. In our corpus,

tidak negates 908 predicates; the total token frequency of the *tidak*-negated predicates is 10,945. Meanwhile, *tak* negates 384 predicate types of the same constructional schemas; the total token frequency of the *tak*-negated predicates is 1,864. The much higher type frequency of *tidak* (i.e., the number of unique negated predicates) suggests that *tidak* has wider applicability, which could also be triggered by its high frequency of occurrence in the corpus. To find the total number of the negated predicates, which is reflected by the total number of rows in each data frame, we can use `nrow()`with the data frame as the input argument, hence `nrow(tak.coll)` and `nrow(tidak.coll)`. To get the total sum of token frequency of these predicates, we can use `sum()` with the `frequency` column as input argument; we access a column of a data frame in R using the dollar sign (`$`), hence `sum(tidak.coll$frequency)` and `sum(tak.coll$frequency)`.

The overall comparison above can be zoomed-in via determining, for each negation, the total number of the negated predicates (i.e., the type frequencies) conforming to the [*me*-X-*kan*], [*di*-X-*kan*], and [*ter*-X-*kan*] schemas.[22] This question relates to the first analytical point mentioned at the end of Section 4.2 regarding the schema-specific association for *tidak* and *tak*. The answer to this question will be used to investigate the hypothesis proposed by Sneddon et al (2010, p. 203) regarding the nuances between *tidak* and *tak* in their usage tendency to negate certain type of predicate schemas, especially *ter*- prefix (see further below).

To find the schema-specific negated collocates, we search within the negated predicates column which predicates begins with each of the three prefixes, ensure that there are no overlapping predicate types for a given schema between *tidak* and *tak*, and count the total number of each set. For that purpose, firstly, we need to introduce another text-processing function in R called `grep()`. `grep()` searches for matches for each element in the input character vector. The output of `grep()` can be of two types: (i) numeric vector(s) indicating the *position index* of the match in the input character vectors (when the `value` argument of `grep()` is set to `FALSE`), or (ii) the match(es) itself as character vectors (when the `value` argument is set to `TRUE`). We briefly illustrate how `grep()` works below (see further Gries, 2009a, pp. 72–73).

When `grep()` is used without regex, it requires two main input-arguments: the search pattern specified in the `pattern` argument and the vector of character strings to be searched for in the `x` argument. Let us illustrate this below with a search-pattern input that is simply `"di"`. We will search this pattern (`"di"`) in the following six words as illustrations, two words for each of the three prefixes: ***me***ndirikan ‘to establish sth.’, ***me***ndiskusikan ‘to discuss sth.’ (for *me*-); ***ter***kalahkan ‘defeatable; lit. be caused to be defeated’, ***ter***hindarkan ‘avoidable; lit be caused to be avoided’ for (*ter*-); ***di***dirikan ‘to be established’, ***di***terjemahkan ‘to be translated’ (for *di*-). The words are intentionally selected to illustrate (i) the output properties of `grep()` and (ii) the importance of specifying the search pattern with regex, especially when dealing with (Indonesian) morphological constructions. To be straightforward, we set the `value` argument to `TRUE` so we can get the matches, rather than the position index of the matches.

```
# create a vector of character strings consisting of six words
predicates <- c("mendirikan", "mendiskusikan", "terkalahkan", "terhindarkan",
"didirikan", "diterjemahkan")

# now use `grep()` to search for "di" character-strings
output <- grep(pattern = "di", x = predicates, value = TRUE)

# check the output
output
## [1] "mendirikan"    "mendiskusikan" "didirikan"     "diterjemahkan"
```

The output tells us something about what `grep()` does. First, feeding `grep()` with a certain input-pattern, such as `"di"`, does not make R know that in fact the users, for instance, wish R to search for `"di"` where `"di"` only occurs at the beginning of the strings (as in ***di****terjemahkan* and ***di****dirikan*), but not in the middle of the strings (as in *men****di****rikan*, *men****di****skusikan*, and *di****di****rikan*). This is the reason why four outputs/elements are returned by `grep()` for the search pattern `"di"`, while we would expect, for our purpose, only two outputs, excluding the two *me-* prefixed words in which `"di"` does occur as part of them. Second, and it is quite related to the first one, `grep()` returns whether the search pattern *occurs at all somewhere* in the input character strings, but does not tell us the *number of its occurrence*; we can see in *didirikan* that the string `"di"` in fact occurs twice: ***didi****rikan*.

Thus, to be more precise in our search pattern and let R "know" what a prefix (or a word) is, we need to design a regex by preceding the prefix with word boundary `"\\b"` that we have seen in Section 4.1, hence `"\\bdi"` for searching *di-* prefixed words from the `predicates` vector. When using regex, the `perl` argument of `grep()` must be set to `TRUE`.

```
# search for 'di-' prefixed predicates with word boundary `\\b`
output <- grep(pattern = "\\bdi", x = predicates, value = TRUE, perl = TRUE)
output
## [1] "didirikan"     "diterjemahkan"
```

The output of `grep()` for a certain search pattern can be fed into `length()` to get the total number of matches found in the character vectors.

```
# search for "di-" prefixed predicates with word boundary `\\b`
output <- grep(pattern = "\\bdi", x = predicates, value = TRUE, perl = TRUE)

# count the length of the output; it should give us two (2)
# 1. First option
length(output)

# 2. Second option
# combine `length()` and `grep()` at once
# don't mix up the closing parentheses for each function!
length(grep(pattern = "\\bdi", x = predicates, value = TRUE, perl = TRUE))
## [1] 2
```

Now, we hope readers can imagine what should be done to answer our question regarding the total number of *me-*, *di-*, and *ter-* prefixed predicates negated with *tidak* and *tak*. The following two code-chunks present the procedures for each *tidak* and *tak*. The first one below shows how to get the character vector of all negated predicates from the columns of each negation data frame shown in Table 2 and Table 3 above.

```
# get the values for the collocate column using the dollar sign ($) from
# each negation data frame. These values then will become a one-dimensional
# character vectors of the negated predicates

# for 'tidak'
tidak.pred <- tidak.coll$predicates

# for 'tak'
tak.pred <- tak.coll$predicates
```

The next chunk shows the searching (with `grep()`) and counting (with `length()`) for the total number of certain morphological forms of the negated predicates. However, before we count the

type frequency of certain form of the predicates, we exclude overlapping predicates between the two negations. That is, we count only unique predicates of certain form that are used with either *tak* or *tidak*, thus no one verb of a given morphological form appears more than once in this dataset (cf. Janda & Lyashevskaya, 2013, pp. 217–218, 221, for similar approach). This procedure is performed to fulfil the independence assumption required by the *chi-square test* that we will use to determine the association of the schema of the predicates with the negations. For that purpose, we use `setdiff()` that requires two vector arguments (`x` and `y`); `setdiff()` determines which elements of `x` that are not found in `y`, thus providing the unique item in `x`.

```
# get the unique types and the number/length of the negated predicates with pa
rticular prefix

# get the predicate types for "tak"
tak.ter <- grep(pattern = "\\bter", x = tak.pred, perl = TRUE, value = TRUE)
tak.di <- grep(pattern = "\\bdi", x = tak.pred, perl = TRUE, value = TRUE)
tak.me <- grep(pattern = "\\bme", x = tak.pred, perl = TRUE, value = TRUE)

# get the predicate types for "tidak"
tidak.ter <- grep(pattern = "\\bter", x = tidak.pred, perl = TRUE, value = TRU
E)
tidak.di <- grep(pattern = "\\bdi", x = tidak.pred, perl = TRUE, value = TRUE)
tidak.me <- grep(pattern = "\\bme", x = tidak.pred, perl = TRUE, value = TRUE)

# get the unique predicates for "tak"
tak.ter.unique <- setdiff(x = tak.ter, y = tidak.ter)
tak.me.unique <- setdiff(x = tak.me, y = tidak.me)
tak.di.unique <- setdiff(x = tak.di, y = tidak.di)

# get the unique predicates for "tidak"
tidak.ter.unique <- setdiff(x = tidak.ter, y = tak.ter)
tidak.me.unique <- setdiff(x = tidak.me, y = tak.me)
tidak.di.unique <- setdiff(x = tidak.di, y = tak.di)

# get the length of the predicates for "tak"
tak.ter.length <- length(tak.ter.unique)
tak.di.length <- length(tak.di.unique)
tak.me.length <- length(tak.me.unique)

# get the length of the predicates "tidak"
tidak.ter.length <- length(tidak.ter.unique)
tidak.di.length <- length(tidak.di.unique)
tidak.me.length <- length(tidak.me.unique)
```

Next, we can inspect whether there are differences in the type frequencies for a given schema across *tak* and *tidak*. In order to determine this, we represent the data in the form of cross-tabulation/contingency table (Gries, 2009b, pp. 127–128) (cf., Table 4 below). The following code-chunk shows how to create a contingency table from the numeric vectors indicating the number of prefix types of the negated predicates.

```
# create the column for each negations
tak.column <- c(tak.me.length, tak.di.length, tak.ter.length)
tidak.column <- c(tidak.me.length, tidak.di.length, tidak.ter.length)

# create contingency table with `cbind()`, which stands for 'column
# binding', from two or more vectors joined as columns
tak.tidak.xtab <- cbind(tak.column, tidak.column)

# inspect the results
tak.tidak.xtab
```

```
##      tak.column tidak.column
## [1,]         29           327
## [2,]         21           246
## [3,]         20            21
```

The first (`[1,]`), second (`[2,]`), and third (`[3,]`) rows indicate, respectively, the predicates with *me-*, *di-*, and *ter-*. We can re-label the columns and provide names for the rows with the following codes.

```
# give names to the row and columns
rownames(tak.tidak.xtab) <- c("me.kan", "di.kan", "ter.kan")
colnames(tak.tidak.xtab) <- c("tak", "tidak")
tak.tidak.xtab
```

Table 4 Type frequencies of the three negated-predicate schemas between *tak* and *tidak*.

	tak	tidak
me.kan	29	327
di.kan	21	246
ter.kan	20	21

The data in Table 4 indicates that predicates with [*ter*-X-*kan*] schema are the least type negated by both *tidak* and *tak*. Meanwhile predicates with [*me*-X-*kan*] schema is the most common across the two negations. The following chunk shows how to generate table of proportion/relative frequencies with `prop.table()` that we have used in Section 4.1.

```
# create proportion with `prop.table()` with the column adds up to 1 (or 100
# if multiplied by 100 to generate percentages) by setting the 'margin'
# argument to '2'; then round the floating points of the proportion values
tak.tidak.xtab.prop <- prop.table(tak.tidak.xtab, margin = 2)
tak.tidak.xtab.prop <- round(tak.tidak.xtab.prop, digits = 2)
tak.tidak.xtab.prop
```

Table 5 Relative type-frequencies of the three negated-predicate schemas between *tak* and *tidak*

	tak	tidak
me.kan	0.41	0.55
di.kan	0.30	0.41
ter.kan	0.29	0.04

A slightly different picture emerges when considering the proportions/relative type-frequencies shown in Table 5. The form *tak* has relatively more types of [*ter*-X-*kan*] predicates to be negated (i.e., 29% types) as compared to *tidak* (i.e., 4% types), even though in the raw type-frequency in Table 4 *tidak* has more number of [*ter*-X-*kan*] predicates than *tak* by just one type. Figure 4 below clearly shows the association of *tak* with [*ter*-X-*kan*] schemas. This kind of plot is called a *mosaic plot* (Gries, 2009b, pp. 129–130, 168; Levshina, 2015, p. 219). We can use `mosaicplot()` from the base R with the cross-tabulation table as the key input-argument.

```
# create mosaic plot
# the `shade = TRUE` parameter produces the colour shading
mosaicplot(x = tak.tidak.xtab, shade = TRUE, cex.axis = 0.75)
```

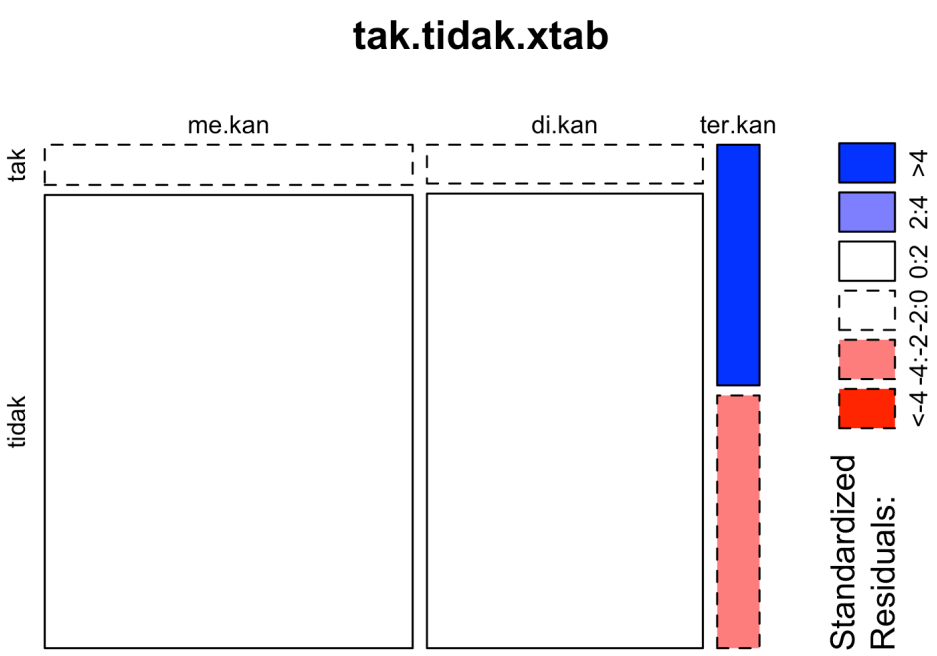

Figure 4 Mosaic plot for the type frequencies of the negated-predicate schemas with *tak* and *tidak*

The area inside the plot reflect the total proportion of the cell-values, rows, and columns in the contingency table. For instance, the bar for [*me*-X-*kan*] is the widest since, overall, this schema has the highest number of predicate types to be negated by the two negations, while [*ter*-X-*kan*] schema has the lowest predicate types. Then, the row-area occupied by *tidak*, especially in the [*me*-X-*kan*] and [*di*-X-*kan*] schemas, are wider than for the [*ter*-X-*kan*] schema, because *tidak* has more number of predicates for these two schemas compared to *tak*.

The dashed boxes in Figure 4 indicates the values of SCHEMAS~NEGATIONS combination whose type frequencies are less frequent than expected under the null-assumption that each negation should have equal ratio of negated predicates conforming to the three schemas. That is, a null-assumption asserts that there is no association between negation forms and the three negated-predicate schemas, in the sense that no difference exists in the ratio of predicates of a given schema to be negated by the two negations (cf., Gries, 2009b, pp. 168–171, for details regarding *observed* vs. *expected* frequencies) (see also Section 4.4.1 below). The clear effect is shown by the higher-than-expected type frequency for the [*ter*-X-*kan*] schema to be negated by *tak* compared to *tidak*. It is indicated by the solid-line rectangle with violet-blue shading, showing positive standardised residuals (see also Table 7 below). In short, the effect suggests that in our database, *tak*, in comparison to *tidak*, prefers to negate verbal predicates with [*ter*-X-*kan*] schema than the other two schemas.

An alternative to the mosaic plot for visualising association between two categorical variables, such as NEGATION and SCHEMAS, is an *association plot*. See the code-chunk below.

```
# association plot with the contingency table input
assocplot(x = tak.tidak.xtab, col = c("black", "grey90"))
```

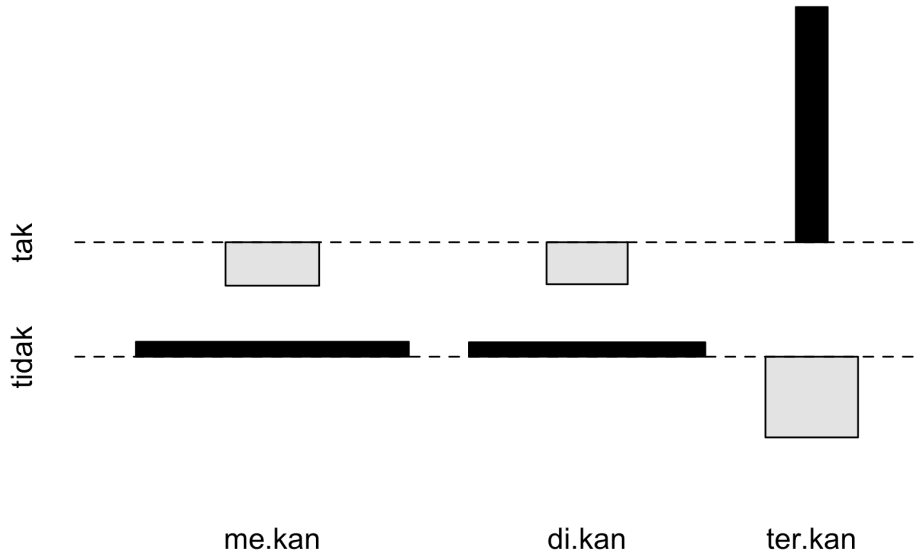

Figure 5 Association plot between the negated-predicate schemas and the negation forms

The black rectangle(s) growing from the dashed line indicates predicate schema(s) (*x*-axis) whose type frequency is overrepresented than expected by chance for a given negation form (*y*-axis). The falling grey rectangle(s) represents the underrepresented combination(s). It is clearly shown that the combination of *tak* and [*ter*-X-*kan*] schema types are more frequent than expected as of *tidak* with [*di*-X-*kan*] schema.

We can further test whether such differences in type frequency of certain negated-predicate schemas for *tak* and *tidak* are due to chance. For this purpose, we use significance testing for contingency table, such as *chi-square test* (Gries, 2013c, pp. 178—189; cf., Janda, 2013, pp. 9–14; Levshina, 2015, p. 210). The *chi-square test* requires that 80% of the expected frequencies are ≥ 5 and all expected frequencies are > 1 (Gries, 2013c, p. 179). To determine the expected frequency for each cell in Table 4, we feed the table into the `chisq.test()` function and retrieve the `expected` elements of the results. The output is shown in Table 6.

```
# perform the chi-square test
# it requires the raw type-frequency contingency table (not percentages table)
taktidak.chisq <- chisq.test(tak.tidak.xtab)

# get the expected frequencies
taktidak.chisq$expected
```

Table 6 Expected frequencies for the contingency table.

	tak	tidak
me.kan	37.530121	318.46988
di.kan	28.147590	238.85241
ter.kan	4.322289	36.67771

Table 6 shows that 5 of the total 6 cells (or 83.33% of the cells) contain expected frequencies larger than five, and all of the expected frequencies are larger than one (Gries, 2013c, p. 179). In that case, the results of the *chi-square test* for the data can be reported.

```
# print the results of chi-square test
taktidak.chisq
##  Pearson's Chi-squared test
```

```
##
## data:  tak.tidak.xtab
## X-squared = 67.763, df = 2, p-value = 1.929e-15
```

The *p*-value[23] of the chi-square test, which is below the conventional significance level of 0.05, suggests that there is a highly significant association between the constructional schemas of the negated predicates and the two negation forms[24]. The mosaic plot (cf., Figure 4) and the association plot (cf., Figure 5) give us visual representations for the direction of the association, namely which negation (dis)prefers which constructional schemas on the basis of the deviation of the observed frequencies from the expected frequencies for each cross-tabulation cell. The direction of the association is also reflected in the values of the so-called *standardised residuals* of a contingency table (cf. Table 7 below), the visualisation of which has been shown in Figure 4. This residual represents which cell has frequency value that significantly deviates from the expected frequency (Levshina, 2015, p. 210). Standardised residuals can be retrieved from the results of a chi-square test as follows.

```
# get the standardised residuals
taktidak.chisq$stdres
```

Table 7 Standardised residuals for the contingency table.

	tak	tidak
me.kan	-2.161547	2.161547
di.kan	-1.842123	1.842123
ter.kan	8.231075	-8.231075

Cells with standardised residuals greater than 1.96 or smaller than -1.96 makes significant contribution to the chi-square/X^2 value at the significance level of 0.05. The values of 2.58 or -2.58 represent residuals for the significance level of 0.01 (Levshina, 2015, p. 221). The positive standardised residuals indicate positive deviation, that is cell-values that are more frequent than expected. The negative residuals are the opposite. Table 7 shows that cells for [*ter*-X-*kan*] schema with *tak* and *tidak* contribute most to the significant association at the significance level of below 0.01, with the combination of *tak* and [*ter*-X-*kan*] schema exhibits positive deviation from the expected frequency.

How can we relate this finding to what Indonesian grammar textbook hypothesises regarding the use of the two negating forms? Sneddon et al (2010, p. 203) states that *tak* "was once obligatory before **ter-** verbs", even though "in modern Indonesian **tidak** is also common in this position" (i.e., in the position of negating *ter*- verbs in general). Our quantitative corpus-based analysis with the chi-square test offers a slightly different nuance. Namely, *tak* negation is not only obligatory before *ter*- verb *once in the past*, but it does maintain its typical usages with this type of verbs in modern Indonesian, at least as framed in the [NEGATION + PREDICATES] schema. Meanwhile *tidak* shows significant dissociation for this type of verbs. Indeed, our result is limited by the corpus for this study, collocational pattern schema (i.e., the [NEGATION + PREDICATES] schema), the regex design, and more importantly by the very specific *ter*- verb schema we focused on (i.e., [*ter*-X-*kan*] schema).

Despite the non-random/significant association between predicates with [*ter*-X-*kan*] schema and *tak* negation, the magnitude/size of this effect is moderate (Cramer's $V = 0.319$) (cf., Gries, 2009b, pp. 173–174; Janda, 2013, pp. 10–11; Levshina, 2015, pp. 208–210)[25]. To sum up, we found that *tak* has preference to negate predicates with [*ter*-X-*kan*] schema in opposition of the schema's dissociation with *tidak* (cf., Figure 5). A chi-square test for

independence indicates that this association cannot be due to chance ($X^2 = 67.76$; $df = 2$; $p_{\text{two-tailed}} < 0.001$); however, this effect is moderate based on the Cramer's V value.

4.4 The verb-specific preferences of tak and tidak

This final analytical section demonstrates that *tak* and *tidak* exhibit verb-specific preferences in their use to negate predicates. Relative preferences for a given negation to negate a set of certain predicates can be another benchmark in distinguishing the use of alternating variant such as *tidak* and *tak* (cf., Gries & Stefanowitsch, 2004; Hilpert, 2006). We argue that such collocational preferences provide a more fine-grained level in characterising the use of *tidak* and *tak*. We will illustrate this idea by performing *Distinctive Collexeme Analysis* on the basis of the whole *token* frequency of the negated-predicate collocates of *tak* and *tidak* that we have retrieved in Section 4.2 and stored in the form of data frames (i.e., `tidak.coll` and `tak.coll`). The snippets of these collocational data have been presented in Table 2 and Table 3 in Section 4.2.

4.4.1 Distinctive Collexeme Analysis: An overview

The method of *Distinctive Collexeme Analysis* (DCA) (cf., Gries & Stefanowitsch, 2004) is a member of a family of quantitative methods called the *Collostructional Analysis* (CA) (cf., Gries, 2012, 2013b; Hilpert, 2014; Stefanowitsch, 2013, 2014; Stefanowitsch & Gries, 2003, 2009, *inter alia*). DCA can be used to reveal differences between two (semantically or functionally) related constructions in terms of statistical association for words that can occur in a slot in the constructions (Stefanowitsch, 2014, p. 226). Among the previous applications of DCA include contrasting verbs that are distinctive for passive vs. active construction in English, preferred infinitival collocates of *will* vs. *be going to*, and verbs strongly attracted to ditransitive vs. prepositional dative constructions (cf., Gries & Stefanowitsch, 2004), or contrasting the English hedges, namely *sort of* vs. *kind of* (Gries & David, 2007).

In our case study, the constructions are negations with a slot for the predicates negated with *tidak* and *tak*. In CA, lexemes (e.g., predicates) occurring in a particular slot (e.g., the negated predicates slot) in a construction (e.g., the Negating Construction) are referred to as the *collexemes* of the construction. With DCA, we aim to uncover whether there are negated collexemes that have significant statistical association with the negated-predicate slot for one of the two Negating Constructions. In other words, we aim to determine whether *tidak* and *tak* exhibit verb-specific preferences that best distinguish between these two (seemingly alternating) negations in their usage to negate predicates. In part, the finding may show us *when* writers (of online news genre in this case) *tend* to choose *tak* over *tidak* to negate certain type of verbal predicates (cf., Hilpert, 2014, p. 398). Table 8 presents a schematic cross-tabulation design required as input for performing DCA.

Table 8 Co-occurrence frequencies required for measuring the distinctiveness/statistical association of a collexeme *X* in two near-synonymous constructions

	Construction_A	Construction_B
collexeme *X*	*a*	*b*
collexeme *ALL-OTHERS*	*c*	*d*

Cell *a* and *b* in Table 8 represent co-occurrence frequencies of a given negated collexeme (i.e., the *collexeme X*) with *Construction_A* and *Construction_B* respectively (cf.,

Levshina, 2015, p. 244, Table 11.1). Cell *c* and *d* represents the remaining token frequencies of the other negated collexemes occurring with *Construction_A* and *Construction_B*. These cross-tabulation data can be evaluated with distributional statistics for statistical significance. CA, including DCA, uses the one-tailed *Fisher Yates Exact* (FYE) test for that purpose (cf., Gries, 2013b, p. 1381). In the remainder of this section, we discuss this underlying statistical technique in more details. As an example, we will determine whether, and to what extent, *terkalahkan* '(can) be defeated; defeatable' is a distinctive negated-collexeme of one of the two negations, namely *tidak* (*Construction_A*) and *tak* (*Construction_B*). Table 9 below shows the input frequency-data for the computation of FYE.

Table 9 Co-occurrence frequency of *terkalahkan* negated with *tidak* and *tak*

	tidak	*tak*	Row totals
terkalahkan	47 (exp: 114.5)	87 (exp: 19.5)	134
others	10,898 (exp: 10,830.5)	1,777 (exp: 1,844.5)	12,675
Column totals	10,945	1,864	12,809

For expository reasons, the observed co-occurrence frequencies for *terkalahkan* with the two Negating Constructions are followed by their corresponding expected frequencies shown in parentheses. The expected frequency of a given co-occurrence represents the frequency that one would expect under the null-hypothesis that the proportion of a collexeme was equally distributed across the two constructions under study (cf., Levshina, 2015, pp. 210–211) (see also Section 4.3 above). In our case, under the null, there should be no difference in the co-occurrence token frequencies of a given predicate, such as *terkalahkan*, to be negated with *tak* and *tidak*.

The expected frequency of *tidak* with *terkalahkan* is computed by multiplying the total frequency of *tidak* (i.e., 10,945) with the total frequency of *terkalahkan* (i.e., 134) and then dividing the multiplication result with the total tokens in the sample (i.e., 12,809), which is the sum of the margin in Table 9. Hence, (10,945∗134)/12,809 = 114.5. The expected frequency for *tak* with *terkalahkan* is calculated similarly; the only difference is the total frequency of *terkalahkan* is now multiplied with the total frequency of *tak* (i.e., 1,864) and then dividied with the total tokens in the sample. Hence, (1,864∗134)/12,809 = 19.5 (cf., Gries, 2013b, p. 1381).

We can submit the data in Table 9 to an FYE test; in R, we can use the `fisher.test()` function[26]. The function requires a 2-by-2 cross-tabulation input of each *observed* co-occurrence frequencies of a collexeme with the constructions (as in Table 9). When using the `fisher.test()`, a one-tailed FYE test can be performed by setting its `alternative` argument to (i) `"greater"` when the observed frequency of the cell of interest (i.e., *a*) is greater than the expected frequency, or to (ii) `"less"` when the observed frequency is less than the expected frequency.[27] Under the null-assumption that the co-occurrence frequency of the collexeme across the two constructions should be equal, the FYE test computes the chance probability of finding a given observed frequency, or even more extreme, given the total frequency of the collexeme under consideration (cf., Janda, 2013, pp. 14–15). Small p_{FYE}-value suggests a low likelihood that the observed frequencies of a collexeme with the two constructions, and its deviation even more, would occur due to chance. If we submit the frequency data in Table 9 into `fisher.test()` and set the `alternative` argument to `"less"` because the co-occurrence frequency of the collexeme with the reference construction[28] (i.e., Construction A) is less frequent than expected, the FYE outputs a very small *p*-value: 8.753e-41 (cf., code-chunk below).

```
# FYE test for the distinctiveness of "terkalahkan"
# create 2-by-2 cross-tabulation matrix
tab <- cbind(c(47, 10898), # for "tidak" column (Construction_A)
             c(87, 1777)) # for "tak" column (Construction_B)

# perform one-tailed FYE
fye <- fisher.test(tab, alternative = "less")

# extract the p-value
fye$p.value
## [1] 8.75284e-41
```

This small *p*-value indicates that *terkalahkan* is highly significantly distinctive for one of the two negations but does not tell us for which one.

The distinctiveness of a collexeme for one of the pair of constructions can be determined from the direction in which the observed co-occurrence frequencies of the collexeme with the two constructions deviate from their expected frequencies. If an observed frequency significantly deviates in a positive direction (i.e., occurring more frequently) than its expected frequency, it represents a *positive* association, or *attraction*, between the collexemes and one of the two constructions; the reverse is an observed frequency occurring less frequently than expected, the situation showing a *negative* association, or *repulsion*, between the collexeme and one of the two constructions (Gries & Stefanowitsch, 2004, p. 103). In DCA, if a collexeme shows attraction for one construction, the given collexeme will be automatically repelled by the other construction (Stefanowitsch, 2005, p. 194, note no. 8); essentially, thus, there are no repelled collexemes in DCA because each collexeme will be assigned into one or the other constructions (Stefanowitsch & Gries, 2009, p. 946).

Table 9 shows that *terkalahkan* co-occurs over four time as often than expected with *tak* and over twice less often than expected with *tidak*. Such deviation, and its statistical significance, indicates that *terkalahkan* is significantly distinctive for *tak*, but significantly not for *tidak*. In a way, this shows us an association between the collexeme and the construction, indicating that the collexeme is a *distinctive collexeme* of the given construction in comparison to the other construction.

In CA, the p_{FYE}-value are $\log_{10}$-transformed into the association strength values called *Coll(ostruction) Str(ength)* (henceforth, CollStr) (Gries, Hampe, & Schönefeld, 2005). This transformation will give *positive* CollStr values showing *attraction*/association and *negative* CollStr values indicating *repulsion*/dissociation. The commonly used threshold in indicating significant collostruction strength is "CollStr > 1.30103", which is equal to $p_{FYE} < 0.05$. Higher cut-off points can also be set, i.e., "CollStr > 2", equal to $p_{FYE} < 0.01$, and "CollStr > 3", equal to $p_{FYE} < 0.001$. These cut-off points are also applicable to indicate significant repulsion when the CollStr values are negative. The higher the CollStr values, the stronger the distinctiveness of a collexeme with a construction, and *vice versa*.

If we repeat this procedure for all the negated collexemes, eventually, DCA allows us to highlight which set of negated collexemes are distinctive for each negation. One could of course repeat *manually* the FYE test in the code-chunk above in R for all the co-occurrence frequencies of the identified negated-predicate collexemes. However, it would be more effective and elegant to wrap this procedure into a programming function that we will turn to in Section 4.4.3 below.

4.4.2 Preparing input data for *Distinctive Collexeme Analysis*

Before performing the *Distinctive Collexeme Analysis* (DCA) for the two negations using the `dist.coll()` R function that we designed (cf., Section 4.4.3), we need to do a bit of data wrangling with the two collocational tables of *tak* (i.e., `tak.coll` data frame) and *tidak* (`tidak.coll`) that we have generated in Section 4.2. It is because we designed the function for a designated input data so that the function can run. The required input-table must consist of three columns (cf., Table 10 below): the first column containing the negated-predicate collexemes and the other two subsequent columns contain the co-occurrence frequencies of the collexemes with *tidak* (i.e., Construction A in the second column) and *tak* (i.e., Construction B in the third column) (cf., Hilpert, 2014, p. 399; Levshina, 2015, pp. 244–245).

Given our `tak.coll` and `tidak.coll` data frames, the first step is to leave out the `percentage` column and to select only the first two columns, namely (i) the `predicates` containing the negated predicates for each negation and (ii) the `frequency` containing the co-occurrence token frequency of each predicate with *tidak* and *tak*. The second step is renaming the second column (i.e., `frequency`) for each table with the corresponding construction labels (i.e., `"tak"` and `"tidak"`). Having one common label (i.e., `predicates`) for one of the two columns, and specifying a unique column label corresponding to the two negations for each of the `frequency` column, allow the merging of the two tables into a single input-table required by our DCA function (i.e., the third step, cf., below). The following code-chunk shows how to perform the first two procedures (selecting columns and renaming a column) for *tidak*.

```
# select the first two columns
tidak.pred <- tidak.coll[, 1:2]

# rename the second, `frequency` column
colnames(tidak.pred)[2] <- "tidak"
```

The chunk below does the same thing as before, now for *tak* data.

```
# select the first two columns
tak.pred <- tak.coll[, 1:2]

# rename the second, `frequency` column
colnames(tak.pred)[2] <- "tak"

# check the first three results for *tak* only
tak.pred[1:3, ]
```

predicates	tak
mengherankan	89
terkalahkan	87
terhindarkan	68

The third step is to merge the above two tables of `tidak.pred` and `tak.pred` into a single table using `merge()` (cf., the code-chunk below). The `x` argument of `merge()` is supplied with the first data frame and the `y` argument is supplied with the second one. We merged `tidak.pred` and `tak.pred` via matching the `predicates` column in each table specified in the `by` argument of `merge()`. We then check the first six rows of the merged table with `head()`.

```
# merge the 'tidak' and 'tak' data
tidaktak <- merge(x = tidak.pred, y = tak.pred, by = "predicates", all = TRUE)
```

```
# `head(tidaktak)` equals `tidaktak[1:6, ]`
head(tidaktak)
```

predicates	tidak	tak
diabaikan	5	1
diacuhkan	3	NA
diadakan	1	NA
diagendakan	2	NA
diajarkan	7	1
diajukan	3	NA

The `NA` values in one of the frequency columns indicate that the given predicates are not found (even once) in the corpus used to be negated with one of the two negations in [*tidak/tak* + PREDICATES] schema. For DCA to run, these `NA` values have to be replaced with zero (`0`). The following code shows how to do that in R. The idea of the code is (i) to subset, with square brackets, all `NA` elements of the `tidaktak` table (hence, the `tidaktak[is.na(tidaktak)]` function-call) and (ii) to replace them with zero (cf., Levshina, 2015, pp. 244–245).

```
# Replace with zero all NAs in the 'tidaktak' table.
tidaktak[is.na(tidaktak)] <- 0
head(tidaktak)
```

Table 10 Co-occurrence frequency of *tidak* and *tak* with the negated predicates

predicates	tidak	tak
diabaikan	5	1
diacuhkan	3	0
diadakan	1	0
diagendakan	2	0
diajarkan	7	1
diajukan	3	0

4.4.3 Performing DCA with the `dist.coll()` function

To perform DCA with in-console output in R, we have designed a function called `dist.coll()`.[29] The function requires three arguments. The first and most important one is the `df` argument, standing for "data frame". The input data frame should contain three columns following the format of the `tidaktak` data frame in Table 10. The first column is the (predicate) collexemes/collocates of the construction. The second and third columns are frequencies of the collexemes with each of the two constructions that do not contain the `NA` values (cf., Section 4.4.2). The second argument is `one.tailed`, which specifies whether to perform one-tailed FYE test (i.e., `TRUE`, the default) or two-tailed (thus, set to `FALSE`). The third argument is `collstr.float.digit`, specifying the floating digits for the Collostruction Strength.

We also designed two other functions to retrieve the top-20 most distinctive collexemes for each of the two constructions: `dist.for.A()` and `dist.for.B()`. For these functions, the first and most important argument is `dca.out`, which is the output table of `dist.coll()`; the second argument, that is `top.n()`, is set by default to `20`, which can be modified by users. The following chunk illustrates the use of `dist.coll()` for performing one-tailed FYE in DCA with our `tidaktak` data.

```
# perform DCA with `dist.col()`
tidaktak.dca <- dist.coll(df = tidaktak, one.tailed = TRUE, collstr.float.digi
t = 2)
```

To retrieve the most distinctive negated collexemes for *tak*, which is the construction B in the input-table for `dist.coll()`, use the `dist.for.B()` function as shown below[30] (cf., Table 11).

```
# retrieve the top-20 most distinctive negated-collexems for 'tak'
dist.for.B(dca.out = tidaktak.dca, top.n = 20)
```

Table 11 Top-20 distinctive negated collexemes for *tak*

	predicates	gloss	tidak	tak	assoc	p.fye	collstr
1	*terkalahkan*	(can) be defeated; defeatable	47	87	A < B	8.75e-41	40.06
2	*terhindarkan*	(can) be avoided; avoidable	20	68	A < B	6.42e-40	39.19
3	*mengherankan*	to (be) astound(ing)	103	89	A < B	2.40e-26	25.62
4	*terelakkan*	(can) be avoided; avoidable	19	44	A < B	3.37e-23	22.47
5	*terpisahkan*	(can) be separated; separatable	23	38	A < B	1.34e-17	16.87
6	*terlupakan*	(can) be forgotten; forgettable	9	29	A < B	1.87e-17	16.73
7	*terbantahkan*	(can) be disputed; disputable	12	30	A < B	1.16e-16	15.93
8	*tergantikan*	(can) be replaced; replaceable	1	20	A < B	3.01e-16	15.52
9	*menyurutkan*	to lessen; abate	59	41	A < B	9.48e-11	10.02
10	*terselamatkan*	(can) be saved	2	11	A < B	3.55e-08	7.45
11	*membuahkan*	to produce; fructify	44	30	A < B	4.19e-08	7.38
12	*tertahankan*	(can) be held; bearable	9	14	A < B	4.08e-07	6.39
13	*diragukan*	to be doubted	100	43	A < B	1.42e-06	5.85
14	*mengenakkan*	cause to be pleased; pleasant	20	17	A < B	4.75e-06	5.32
15	*terbayangkan*	(can) be imagined; imaginable	12	13	A < B	1.18e-05	4.93
16	*terelakan*	(can) be let go	1	6	A < B	5.78e-05	4.24
17	*terperikan*	(can) be expressed; expressible	0	5	A < B	6.50e-05	4.19
18	*mematahkan*	to break	1	4	A < B	1.98e-03	2.70
19	*termaafkan*	(can) be forgiven; forgivable	1	4	A < B	1.98e-03	2.70
20	*tergoyahkan*	(can) be shaken	10	8	A < B	2.21e-03	2.65

The `assoc` column with the value `"A < B"` indicates that the observed co-occurrence frequency of a given predicate is more frequent than expected with *tak* (i.e., `"B"`), but less frequent with *tidak* (i.e., `"A"`). The `collstr` column shows the CollStr values derived from the negative $\log_{10}$-transformed $p_{\text{ONE-TAILED}}$-value of the FYE test in `p.fye` column. The CollStr values in Table 11 show the magnitude of distinctiveness of the predicates with *tak*. In relation to *tidak*, these CollStr values can also be interpreted as the degree of repulsion, or indistinctiveness, of the predicates with *tidak* (cf., Section 4.4.1 for interpreting results of DCA). Moreover, the CollStr values for *tak*, at least in the first eight rows, are (much) higher than the CollStr for the top collexemes for *tidak* (cf., Table 12 for *tidak* below). This indicates much stronger distinctiveness of these predicates with the given construction (i.e., *tak*).

As can further be seen from Table 11, 70% of the top-20 most distinctive negated collexemes for *tak* is of the [*ter*-X-*kan*] schema. In contrast, none of the top-20 distinctive negated collexemes for *tidak* in Table 12 below occur in the [*ter*-X-*kan*] schema; in fact, the [*ter*-X-*kan*] collexemes in Table 11 are those strongly indistinctive for *tidak*. The majority (i.e., 65%) of the top-20 distinctive negated collexemes of *tidak* are in the active voice schema with

[*me*-X-*kan*]. The following code shows the use of `dist.for.A()` to output the top-20 most distinctive collexemes for *tidak* (i.e., the construction A).

```
# retrieve the top-20 most distinctive negated-collexems for 'tidak'
dist.for.A(dca.out = tidaktak.dca, top.n = 20)
```

Table 12 Top-20 distinctive negated collexemes for *tidak*

	predicates	gloss	tidak	tak	assoc	p.fye	collstr
1	*melakukan*	to do	669	44	A > B	4.93e-13	12.31
2	*menimbulkan*	to cause to emerge	271	12	A > B	1.23e-08	7.91
3	*dilakukan*	to be done	314	17	A > B	2.83e-08	7.55
4	*menggunakan*	to u(tili)se	299	20	A > B	2.35e-06	5.63
5	*memberikan*	to give	506	51	A > B	6.47e-05	4.19
6	*melaksanakan*	to carry out	70	2	A > B	1.03e-03	2.99
7	*menaikkan*	to raise	52	1	A > B	2.37e-03	2.62
8	*dibenarkan*	to be justified	71	3	A > B	3.57e-03	2.45
9	*ditemukan*	to be found	229	23	A > B	6.33e-03	2.20
10	*merugikan*	to adverse/harm	81	5	A > B	9.72e-03	2.01
11	*menjalankan*	to carry out	60	3	A > B	1.30e-02	1.89
12	*mendapatkan*	to receive	286	33	A > B	1.59e-02	1.80
13	*dimanfaatkan*	to be made use of	65	4	A > B	2.04e-02	1.69
14	*menjanjikan*	to be promising	24	0	A > B	2.29e-02	1.64
15	*dilaksanakan*	to be carried out	71	5	A > B	2.66e-02	1.58
16	*diperbolehkan*	to be allowed	133	13	A > B	2.82e-02	1.55
17	*menyebabkan*	to cause	34	1	A > B	2.82e-02	1.55
18	*menetapkan*	to determine	22	0	A > B	3.13e-02	1.50
19	*memberatkan*	to (be) burden(some)	33	1	A > B	3.22e-02	1.49
20	*dijadikan*	to be used as	51	3	A > B	3.54e-02	1.45

The results of DCA refine our previous finding in Section 4.3 regarding the association of *tak* with the static-passive [*ter*-X-*kan*] schema. In this paper, we showed that not only *tak* has relatively higher number of [*ter*-X-*kan*] predicates (i.e., the type frequency of the schema) than expected by chance in comparison to the other two schemas and to *tidak*, but also it has strong association with a set of [*ter*-X-*kan*]-specific predicates in its usage occurrences (i.e., *tak*'s token frequencies) with these predicates. The distinctive negated collexemes in Table 11 and Table 12 offer more detailed usage-based data on how the two negations are used in a set of statistically-based, entrenched collocations in a natural language setting, especially in a written mode such as online news. In turn, the data can be instrumental, among others, for teaching Indonesian to foreign learners in helping them to reach a native-like performance when using a negation in its distinctive collocational patterns.

5 CONCLUSION

In this paper, we have illustrated the use of R in conducting a series of quantitative corpus-based analyses on Indonesian Negating Constructions. We found that *tidak* is the most frequent form occurring in a 17,942,268 million-word online news corpus (Section 4.1). Then, we identified significant differences, but with moderate effect, between the two most frequent negations, namely *tidak* and *tak*, in terms of the type frequencies of three negated-predicate schemas,

namely [*me*-X-*kan*], [*di*-X-*kan*], and [*ter*-X-*kan*] (Section 4.3). The clear effect emerging from the data is the significant association between *tak* and the [*ter*-X-*kan*] schema, which is relatively not the case for *tidak* (cf., e.g., Figure 5). Our corpus-based finding provides a quantitative nuance to Sneddon et al's (2010, p. 203) hypotheses, stating that (i) *tidak* is common for negating *ter-* predicates in present-day Indonesian, and (ii) the preference of *tak* for *ter-* verbs is a past conventional usage; what we found instead is that while *tidak* does negate predicates immediately following it in [*ter*-X-*kan*] schema, this usage is relatively more associated with *tak* even in the present-day Indonesian. Yet, practical relevance of such effect might not be that substantial (note the moderate Cramer's *V* measure of this analysis). Lastly, we have introduced a quantitative technique called *Distinctive Collexeme Analysis* (DCA) and demonstrated how it can be computed in R (Section 4.4). We applied DCA to measure a more fine-grained level of usage differentiating *tidak* and *tak*, namely their verb-specific preferences. The results of DCA have shown that *tak* in its usage tokens predominantly collocates with certain [*ter*-X-*kan*] predicates, which are in turn strongly repelled by *tidak* (Table 11); this can enrich our initial finding on the association of the schema with *tak*. We also pointed out a potential implication of these distinctive collocational data for foreign learners of Indonesian.

We acknowledge, however, that our results are limited, in terms of the corpus, focused constructional schemas, and the design of the searching patterns (i.e., regexes) in extracting the collocations. We hope further studies can replicate and broaden our work with different corpus genre of potentially larger size and with wider analytical scopes. A potential follow-up from this study, especially in relation to the findings of DCA, is to determine the extent to which the occurrences of particular predicates in a negating construction are predictive for the presence of *tak* versus *tidak* as the negating forms, and *vice versa*; this can be measured using the Delta P (ΔP) statistics (cf., Gries, 2013a; Levshina, 2015, pp. 232, 234). After all, this paper reveals only the tip of the R-iceberg, especially for performing quantitative corpus linguistics. Be that as it may, in the remainder of this section, we intend to summarise our thoughts about, at least, two reasons why it is worth investing the time to explore the world of R for corpus linguistics.

The first reason is the nature of R as a programming language. As a *programming* language, R allows its users to create their own programs/functions to perform whatever operations they could imagine in relation to processing and analysing corpus data. This versatility of R means that users are no longer tied to the limitations of some other commercial or free corpus linguistic tools for performing certain tasks, in the sense that when a given corpus linguistic software cannot do X, neither can the users. Readers can relate this argument, for instance, to what we did for performing DCA, and to the other all-in-one statistical and text-processing analyses showcased in this paper. The second reason is the well-tailored ecology offered by RStudio and R Markdown notebook for the unified and more reproducible research workflows with R. These run the gamut of importing the (raw) data, (pre-)processing the data, analysing the data into insights, and finally communicating the results into a report, as embodied in this paper. In our view, these integrated features of R and its powerful computational tools, among others, take us to the cutting-edge of data science in corpus linguistics.

REFERENCES

Anthony, L. (2014). AntConc (Version 3.4.3). Tokyo, Japan: Waseda University. Retrieved from http://www.laurenceanthony.net

Arka, I. W. (2010, August). *Dynamic and stative passives in Indonesian & their computational implementation*. Paper presented at the MALINDO Workshop, Jakarta: Paper.

Baayen, R. H. (2008). *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge, UK ; New York: Cambridge University Press.

Biemann, C., Heyer, G., Quasthoff, U., & Richter, M. (2007). The Leipzig Corpora Collection: Monolingual corpora of standard size. In M. Davies, P. Rayson, S. Hunston, & P. Danielsson (Eds.), *Proceedings of the Corpus Linguistics Conference*. University of Birmingham, UK. Retrieved from http://ucrel.lancs.ac.uk/publications/CL2007/paper/190_Paper.pdf

Desagulier, G. (2017). *Corpus linguistics and statistics with R: Introduction to quantitative methods in linguistics*. New York, NY: Springer Berlin Heidelberg.

Diessel, H. (2015). 14. Usage-based construction grammar. In E. Dabrowska & D. Divjak (Eds.), *Handbook of Cognitive Linguistics* (pp. 296–322). Berlin ; Boston: De Gruyter Mouton.

Dinakaramani, A., Rashel, F., Luthfi, A., & Manurung, R. (2014). Designing an Indonesian part of speech tagset and manually tagged Indonesian corpus. In *2014 International Conference on Asian Language Processing (IALP)* (pp. 66–69). doi:10.1109/IALP.2014.6973519

Flanagan, J. (2017). Reproducible research: Strategies, tools, and workflows. *Studies in Variation, Contacts and Change in English*, *19*. Retrieved from http://www.helsinki.fi/varieng/series/volumes/19/flanagan/

Goldberg, A. E. (1995). *Constructions: A construction grammar approach to argument structure*. Chicago: University of Chicago Press.

Goldberg, A. E. (2006). *Constructions at work: The nature of generalization in language*. Oxford: Oxford University Press.

Goldhahn, D., Eckart, T., & Quasthoff, U. (2012). Building large monolingual dictionaries at the Leipzig Corpora Collection: From 100 to 200 languages. In *Proceedings of the 8th Language Resources and Evaluation Conference (LREC) 2012* (pp. 759–765). Istanbul. Retrieved from http://www.lrec-conf.org/proceedings/lrec2012/pdf/327_Paper.pdf

Gries, S. T. (2009a). *Quantitative corpus linguistics with R: A Practical Introduction*. New York: Routledge.

Gries, S. T. (2009b). *Statistics for linguistics with R: A practical introduction*. Berlin: Mouton de Gruyter.

Gries, S. T. (2012). Collostructions. In P. J. Robinson (Ed.), *The Routledge encyclopedia of second language acquisition* (pp. 92–95). London: Routledge.

Gries, S. T. (2013a). 50-something years of work on collocations: What is or should be next …. *International Journal of Corpus Linguistics*, *18*(1), 137–166. doi:10.1075/ijcl.18.1.09gri

Gries, S. T. (2013b). Corpus linguistics: Quantitative methods. In C. A. Chapelle (Ed.), *The Encyclopedia of Applied Linguistics* (Vols. 1–10, pp. 1380–1385). Chichester, West Sussex, UK: Blackwell Publishing Ltd. doi:10.1002/9781405198431.wbeal0258

Gries, S. T. (2013c). *Statistics for linguistics with R: A practical introduction* (2nd). Berlin: Mouton de Gruyter.

Gries, S. T. (2014). *Coll.Analysis 3.5. A script for R to compute perform collostructional analysis*. Retrieved from http://www.linguistics.ucsb.edu/faculty/stgries/teaching/groningen/readme.txt

Gries, S. T., & David, C. V. (2007). This is kind of / sort of interesting: Variation in hedging in English. *Towards Multimedia in Corpus Studies*, *2*. Retrieved from http://www.helsinki.fi/varieng/journal/volumes/02/gries_david/

Gries, S. T., & Stefanowitsch, A. (2004). Extending collostructional analysis: A corpus-based perspective on 'alternations'. *International Journal of Corpus Linguistics*, *9*(1), 97–129.

Gries, S. T., Hampe, B., & Schönefeld, D. (2005). Converging evidence: Bringing together experimental and corpus data on the association of verbs and constructions. *Cognitive Linguistics*, *16*(4), 635–676.

Hilpert, M. (2006). Distinctive collexeme analysis and diachrony. *Corpus linguistics and linguistic theory*, *2*(2), 243–256.

Hilpert, M. (2014). Collostructional analysis: Measuring associations between constructions and lexical elements. In D. Glynn & J. A. Robinson (Eds.), *Corpus methods for semantics: Quantitative studies in polysemy and synonymy* (pp. 391–404). Amsterdam: John Benjamins Publishing Company.

Janda, L. A. (2013). Quantitative methods in *Cognitive Linguistics*: An introduction. In L. A. Janda (Ed.), *Cognitive linguistics: The quantitative turn* (pp. 1–32). Berlin: Mouton de Gruyter.

Janda, L. A., & Lyashevskaya, O. (2013). Semantic profiles of five Russian prefixes: *Po-*, *s-*, *Za-*, *Na-*, *Pro-*. *Journal of Slavic Linguistics*, *21*(2), 211–258. doi:10.1353/jsl.2013.0012.

Kroeger, P. (2014). External negation in Malay/Indonesian. *Language*, *90*(1), 137–184. doi:10.1353/lan.2014.0000.

Larasati, S. D., Kuboň, V., & Zeman, D. (2011). Indonesian Morphology Tool (MorphInd): Towards an Indonesian Corpus. In *Systems and Frameworks for Computational Morphology* (pp. 119–129). Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-23138-4_8.

Levshina, N. (2015). *How to do Linguistics with R: Data exploration and statistical analysis*. John Benjamins Publishing Company.

R Core Team. (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.R-project.org/

Sanchez, G. (2013). *Handling and processing strings in R*. Berkeley: Trowchez Editions. Retrieved from http://www.gastonsanchez.com/Handling and Processing Strings in R.pdf

Sneddon, J. N. (2006). *Colloquial Jakartan Indonesian*. Canberra, Australia: Pacific Linguistics, Research School of Pacific and Asian Studies, The Australian National University.

Sneddon, J. N., Adelaar, A., Djenar, D. N., & Ewing, M. C. (2010). *Indonesian reference grammar* (2nd). Crows Nest, New South Wales, Australia: Allen & Unwin.

Stefanowitsch, A. (2005). The function of metaphor: Developing a corpus-based perspective. *International Journal of Corpus Linguistics*, *10*(2), 161–198. doi:10.1075/ijcl.10.2.03ste

Stefanowitsch, A. (2013). Collostructional analysis. In T. Hoffmann & G. Trousdale (Eds.), *The Oxford handbook of Construction Grammar*. Oxford: Oxford University Press. doi:10.1093/oxfordhb/9780195396683.013.0016

Stefanowitsch, A. (2014). Collostructional analysis: A case study of the English *into*-causative. In T. Herbst, H.-J. Schmid, & S. Faulhaber (Eds.), *Constructions collocations patterns*. Berlin ; Boston: Walter De Gruyter, GmbH.

Stefanowitsch, A., & Gries, S. T. (2003). Collostructions: Investigating the interaction of words and constructions. *International Journal of Corpus Linguistics*, *8*(2), 209–243.

Stefanowitsch, A., & Gries, S. T. (2009). Corpora and grammar. In A. Lüdeling & M. Kytö (Eds.), *Corpus linguistics: An international handbook* (Vol. 2, pp. 933–951). Berlin: Mouton de Gruyter.

Wickham, H., & Grolemund, G. (2017). *R for Data Science*. Canada: O'Reilly. Retrieved from http://r4ds.had.co.nz/

---

[1] The writing of this paper is supported by (i) the *Monash International Postgraduate Research Scholarships* (MIPRS) and the *Monash Graduate Scholarships* (MGS) (awarded to Gede Primahadi Wijaya Rajeg) as well as by the *MonARCH High Performance Computing* cluster at Monash University, Australia; and by (ii) the *Indonesian Endowment Fund for Education* (LPDP) (awarded to Karlina Denistia).

[2] *Data science* here is not intended to be a buzz word, but a scientific enterprise about how "to turn raw data into understanding, insight, and knowledge" (Wickham & Grolemund, 2017, p. ix).

[3] It should also be mentioned that there is a great free corpus linguistic software, namely *AntConc*, that can do most of the core corpus linguistic analyses (e.g., creating concordances, frequency lists, collocational analyses, etc.) (cf., Anthony, 2014).

[4] The term *construction* here is understood as form-meaning pairing as in the *Construction Grammar* approach (cf., Diessel, 2015; Goldberg, 1995, 2006, *inter alia*).

[5] The text is a chunk of a short story entitled *Darojat dan Istrinya* 'Darojat and his wife' written by Budi Darma. The story was published in *Jawa Pos* and we scrapped the text with R from https://lakonhidup.wordpress.com/2016/07/17/darojat-dan-istrinya/ on 3 September 2016 at 12.19 PM.

[6] Download *Indonesian POS Tagger* from https://github.com/andryluthfi/indonesian-postag. For running the POS Tagger, users are also required to install *MorphInd* (Larasati, Kuboň, & Zeman, 2011). Details information on *MorphInd*, including how to download and install it, can be found at http://septinalarasati.com/morphind/.

[7] https://cran.r-project.org

[8] https://www.r-project.org/conferences.html
[9] https://journal.r-project.org
[10] https://www.r-bloggers.com
[11] https://stackoverflow.com
[12] The R code-chunk is indicated with a grey-shaded background with `Courier` font family for the text. The gold-brown texts following a hashtag "#" are the authors' comments on the operations that the codes perform; R will not execute comments. The output of an operation will be marked with a double hashtag ## rendered in black fonts.
[13] To see the required argument(s) of a function and how it is used, in the R console, one can type question mark (?) followed by the name of the function. For instance, typing `?round()` will open up a window showing the documentation and details of the `round()` function.
[14] Download RStudio from https://www.rstudio.com/products/rstudio/download/. R has to be installed before installing RStudio.
[15] R Markdown is an R package that can be installed by running `install.packages("rmarkdown")` (but cf. Wickham & Grolemund, 2017, p. 424). For details, see also http://rmarkdown.rstudio.com/index.html.
[16] http://rmarkdown.rstudio.com/r_notebooks.html
[17]For introductions to working with R Markdown for MS Word output, see http://rmarkdown.rstudio.com/word_document_format.html and http://rmarkdown.rstudio.com/articles_docx.html. Readers may also explore Yihui Xie's free online-book on bookdown package for producing book via R Markdown (https://bookdown.org/yihui/bookdown/).
[18] The R Markdown file is made open-access with CC BY-NC-SA 4.0 licence on our *figshare* webpages: https://figshare.com/articles/Working_with_a_linguistic_corpus_using_R_An_introductory_note_with_Indonesian_Negating_Construction/5873256.
[19] Download Leipzig Corpora from http://wortschatz.uni-leipzig.de/en/download/.
[20] *Perl* is another kind of programming language.
[21] There is a faster alternative to the combination of `gregexpr()` and `regmatches()`. It is the `str_extract_all()` function from the `stringr` package, which can be installed as part of the coherent family of packages called the `tidyverse` (cf., Wickham & Grolemund, 2017, for a comprehensive introduction to the `tidyverse` packages; further details of the `tidyverse` can also be found at https://www.tidyverse.org). We do not use `str_extract_all()` and any other functions from the core `tidyverse` packages to familiarise readers with what the base R can offer to the tasks at hand. Once the readers feel comfortable with the base R, we do encourage readers to advance to the `tidyverse`, which has been very popular among the majority of the R communities.
[22] The underlying design for this analysis is adapted from a study by Janda and Lyashevskaya (2013) that contrasts the semantics of five aspectual prefixes in Russian according to the number of the verbs tagged with a set of lexico-semantic classes that are prefixed with one of the five aspectual prefixes.
[23] The non-scientific equivalent of the *p*-value of 1.929e-15 is 0.000000000000001929177.
[24] A stricter practice is to use `fisher.test()` rather than `chisq.test()` when none of the expected frequencies in the table are larger than five. When we submit the `tak.tidak.xtab` data into `fisher.test`, the result is still significant as produced by the chi-square test since the *p*-value is below the standard threshold of 0.05 (i.e. p = 3.683827e-10). Run the following line in R console `fisher.test(tak.tidak.xtab)$p.value` to get the p-value.
[25] Cramer's *V* value is a measure of the size of effect found in a contingency table larger than 2-by-2 table, as in the case of our negation data (Levshina, 2015, p. 217). Cramer's *V* value is not affected by the sample size, unlike the *p*-value. The range of the Cramer's *V* value spans from 0 (no effect/no correlation) to 1 (large effect/perfect correlation) (cf., Levshina, 2015, p. 209). The code for the Cramer's *V* value in this paper is `sqrt(taktidak.chisq$statistic/sum(tak.tidak.xtab)*(min(dim(tak.tidak.xtab))-1))` (cf. Gries, 2013c, p. 186).
[26] See Gries (2013b, p. 1381) for the use of the `dhyper()` function in R as an alternative to perform a one-tailed FYE test.
[27] See the *Youtube* video featuring Laura A. Janda entitled *The use of statistics in Cognitive Linguistics* on her presentation slide at 28:39-28:45 minutes (https://www.youtube.com/watch?v=lCJw-Qd3ZZg).
[28] See Levshina (2015, p. 245) for the idea of reference construction in DCA, as well as Gries (2013b, p. 1381) for how the frequency deviation in a reference construction affects the computation of the one-tailed FYE.

[29] The R code for our `dist.coll()` function is embedded in the R Markdown document of this paper. The core FYE computation in `dist.coll()` is adapted from Levshina's (2015, p. 245) `pv.Fisher.collstr()`. We customised our function with options to compute one- or two-tailed FYE by specifying the value of the `alternative` argument for the embedded `fisher.test()` function; in Levshina's `pv.Fisher.collstr()`, the `alternative` argument is set to the default `"two.sided"` test. In addition, our `dist.coll()` function returns a data-frame object (cf. Table 11 and Table 12, minus the gloss) that can be directly processed in R console. There is also a well-known R script, namely *Coll.Analysis 3.5* (Gries, 2014), that can perform all of the Collostructional Analysis family of methods and that prints out the results either into a plain text format or into the R console screen.

[30] The gloss column is not part of the output because it is manually added by the authors.