Text Preprocessing

Promothesh Chatterjee*

Overview:

We will try to understand how to process text data for analysis. We will understand concepts such as:

- Tokenization
- Stemming
- Lemmatization
- Part of Speech Tagging
- Document Term Matrix

How is Text Data Different from Numeric Data?

You might think that once somehow we are able to represent text quantitatively, rest should be pretty much the same as regular data analysis. Unfortunately, you will find out that the way text is represented often makes it difficult to do conventional data analysis.

Let's look at a hotel review from a consumer for the purpose of illustration.

Consumer Review: *This Residence Marriott is not fancy. It was clean, good service and suite style rooms; however, it's a little older than some Residence Marriotts's I've stayed at and probably could use an update soon.*

Our aim here is to represent this sentence numerically so that most data science algorithms can read it. Textual data is mostly a collection of sentences and paragraphs which can be broken down into words and punctuations. The logical way is to parse this data into its most granular element - tokens. The easiest way to tokenize is to split the text into words. However, there are other ways to tokenize a text depending on the aim of analysis. There are many packages which can perform tokenization, we will look at R package, tokenizers to understand tokenization in greater details (later on we will mostly use R package Quanteda).

The above sentence may be parsed like this into tokens:

{This} {Residence} {Marriott} {is} {not} {fancy} {.} {It} {was} {clean} {,} {good} {service} {and} {suite} {style} {rooms} {;} {however} {,} {it's} {a} {little} {older} {than} {some} {Residence} {Marriotts's} {I've} {stayed} {at} {and} {probably} {could} {use} {an} {update} {soon} {.}

Note, that punctuation too, are part of tokens (in preprocessing we typically remove the punctuation) along with words. Let's take a look at different ways of tokenization using R package, tokenizers. Later we will look at tokenization using R package Quanteda that we use very frequently in our class.

```
library(tokenizers)
Review<-paste0("This Residence Marriott is not fancy.
               It was clean, good service and suite style rooms.
               It's a little older than some Residence Marriotts's.
               Probably could use an update soon.")
```

```
tokenize_words(Review)#### Regular tokenization
```

Implementing Text Preprocessing in R

```
## [[1]]
##  [1] "this"        "residence"   "marriott"    "is"          "not"
##  [6] "fancy"       "it"          "was"         "clean"       "good"
## [11] "service"     "and"         "suite"       "style"       "rooms"
## [16] "it's"        "a"           "little"      "older"       "than"
## [21] "some"        "residence"   "marriotts's" "probably"    "could"
## [26] "use"         "an"          "update"      "soon"
```

R package textclean provides many functions for replacing dates, emojis, emoticons,etc. Let's replace the contractions in the consumer review such as "it's" with "it is" and "I've" with "I have" using function replace_contraction from textclean.

```
library(textclean)
Review<-replace_contraction(Review, contraction.key = lexicon::key_contractions)
```

Take a look at some of these functions here: https://github.com/trinker/textclean

Stopwords

Stopwords are commonly occurring words that do not add meaning to the context, such as articles, prepositions, punctuations, hyphens, numbers and symbols otherwise they will also show up in the data matrix. Different packages have implemented different stopword lists; these can be general or domain specific. Popular stopword lists are: the SMART (System for the Mechanical Analysis and Retrieval of Text) (Lewis et al. 2004), English Snowball stop word list (Martin F. Porter 2001), Stopwords ISO collection. For now, we will use the stopwords dictionary implemented in library tokenizers.

```
tokenize_words(Review, stopwords = stopwords::stopwords("en")) ## Tokenization with stop words
```

```
## [[1]]
##  [1] "residence"   "marriott"    "fancy"       "clean"       "good"
##  [6] "service"     "suite"       "style"       "rooms"       "little"
## [11] "older"       "residence"   "marriotts's" "probably"    "use"
## [16] "update"      "soon"
```

Tokens can be other units of text not just words. For instance, we can tokenize at the level of characters, words, sentences, lines, paragraphs, and n-grams.

Let's see how we can tokenize at the level of characters:

```
tokenize_characters("Hello, World!")
```

```
## [[1]]
##  [1] "h" "e" "l" "l" "o" "w" "o" "r" "l" "d"
```

We often need to split texts into sentences or paragraphs prior to tokenizing into other forms. Library tokenizers can be used for these too!

```
tokenize_sentences(Review)
```

```
## [[1]]
## [1] "This Residence Marriott is not fancy."
## [2] "It was clean, good service and suite style rooms.                it is a little older than some
## [3] "Probably could use an update soon."
```

```
tokenize_paragraphs(Review)
```

```
## [[1]]
## [1] "This Residence Marriott is not fancy.                 It was clean, good service and suite styl
```

We will look at n-grams in a subsequent section.

For more details, check out:

https://cran.r-project.org/web/packages/tokenizers/vignettes/introduction-to-tokenizers.html

Stemming

Stemming typically cuts words to their root words. Consider words, 'less', 'lesser', 'lessen', 'lessor'. These words may be reduced to a common stem "less". Note though, the word 'lessor' refers to 'One who grants or gives a lease' and as such this stemming would not be correct. Stemming is necessary because different forms of the same words increase the sparsity of word matrices (more on this when we talk about the curse of dimensionality). Some commonly used stemming algorithms are Porter's Stemmer, Lancaster Stemmer (we used this for the above example), SnowBall Stemmer. In the R package, tokenizers, default stemming is done with the help of SnowballC package.

```
tokenize_word_stems(Review) ## SnowballC package
```

```
## [[1]]
##  [1] "this"     "resid"    "marriott" "is"       "not"      "fanci"
##  [7] "it"       "was"      "clean"    "good"     "servic"   "and"
## [13] "suit"     "style"    "room"     "it"       "is"       "a"
## [19] "littl"    "older"    "than"     "some"     "resid"    "marriott"
## [25] "probabl"  "could"    "use"      "an"       "updat"    "soon"
```

Another popular word-stemmer that preserves punctuation and separates common English contractions is the Penn Treebank tokenizer. R package, tokenizers, can implement that as well.

```
tokenize_ptb(Review) # Penn Treebank tokenizer
```

```
## [[1]]
##  [1] "This"         "Residence"    "Marriott"     "is"           "not"
##  [6] "fancy."       "It"           "was"          "clean"        ","
## [11] "good"         "service"      "and"          "suite"        "style"
## [16] "rooms."       "it"           "is"           "a"            "little"
## [21] "older"        "than"         "some"         "Residence"    "Marriotts's."
## [26] "Probably"     "could"        "use"          "an"           "update"
## [31] "soon"         "."
```

Lemmatization

Unlike stemming, lemmatization reduces words to their linguistically correct base words or lemmas. For instance, consider these words: 'bad', 'worse', 'worst'. Lemmatization uses vocabulary and morphological analysis to come up with the correct lemma- 'bad'. This would obviously not be captured by a stemmer. Spacy in Python's NLTK and SpacyR in R can be used for lemmatization. In R, SpacyR is implemented using cleanNLP package but it can be very resource intensive. A simpler implementation is from library textstem.

```
library(textstem)
lemmatize_words(c("bad", "worse","worst"))
```

```
## [1] "bad" "bad" "bad"
```

POS Tagging

POS tagging refers to Part-of-Speech(POS) tagging. POS tagging enables us to label the words in a sentence appropriately with different parts of speech such as nouns, pronouns, verbs, adverbs, adjectives, conjunctions etc. While this doesn't help with reducing sparsity and dimensions, but can be useful in understanding the context. Penn Treebank is a popular tag set.

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	*+,%, &*	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

We can use library cleanNLP for lemmatization and POS tagging, but this is extremely resource intensive and in real life situations, it is not always worth the trade-off. The code below is just for demonstration but practically often we will stop text preprocessing after stemming.

```
library(cleanNLP)
 cnlp_init_udpipe()# udpipe is another package which we use as backend
 cnlp_annotate("This Residence Marriott is not fancy.")
```

```
## $token
##   doc_id sid tid     token token_with_ws     lemma  upos xpos
## 1      1   1   1      This          This       this   DET   DT
## 2      1   1   2 Residence     Residence  residence  NOUN   NN
## 3      1   1   3  Marriott      Marriott   Marriott PROPN  NNP
## 4      1   1   4        is            is         be   AUX  VBZ
## 5      1   1   5       not           not        not  PART   RB
## 6      1   1   6     fancy         fancy      fancy   ADJ   JJ
## 7      1   1   7         .             .          . PUNCT    .
##                                                  feats tid_source relation
## 1                              Number=Sing|PronType=Dem          3      det
## 2                                           Number=Sing          3 compound
## 3                                           Number=Sing          6    nsubj
## 4 Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin          6      cop
## 5                                                 <NA>          6   advmod
## 6                                          Degree=Pos          0     root
## 7                                                 <NA>          6    punct
##
## $document
```

```
##   doc_id
## 1      1
##
## attr(,"class")
## [1] "cnlp_annotation" "list"
```

Representing Text As Dataset

Now the question is how to represent these tokens as an array or a matrix (that being the typical way to represent datasets for statistical analysis). In a typical quantitative data matrix, we have observations in rows and features/variables in columns. One way to represent this is using a document term matrix or DTM. In a document term matrix, we have documents in rows (for instance, in our example, different reviews can be in rows or even different lines) and tokens/words in columns.

DATA MATRIX

Features/ Variables

Observations	X1	X2	X3	.	.	.	Xp	Y
1								
2								
3								
4								
.								
.								
N								

DOCUMENT TERM MATRIX

Tokens/ Words

Documents	T1	T2	T3	.	.	.	Tp	Y
1								
2								
3								
4								
.								
.								
N								

Please note that p dimensions of DTM are generally much greater than in a regular data matrix. Thus, we can organize text as a matrix with nxp dimensions. For instance, the above hotel review can be converted into a Document Term Matrix (R package Quanteda calls it Document Feature Matrix instead) like this:

For sake of demonstration of the concept of DTM, I have split the review into 3 documents

text1= “This Residence Marriott is not fancy.”

text2= “It was clean, good service and suite style rooms;”

text3= “However, it’s a little older than some Residence Marriotts’s I’ve stayed at and probably could use an update soon.”

```
library(quanteda) # this is a popular text package which we will use a lot
a<-"This Residence Marriott is not fancy."
b<-"It was clean, good service and suite style rooms;"
c<-"However, it's a little older than some Residence Marriotts's
    I've stayed at and probably could use an update soon."

hotel_demo<-c(a,b,c)
hotel_demo_token <- quanteda::tokens(hotel_demo,
                       remove_numbers = TRUE, remove_punct = TRUE,
                       remove_symbols = TRUE, remove_hyphens = TRUE)
# instead of using library
# tokenizers we are using the function tokens from quanteda

hotel_demo_dfm<-dfm(hotel_demo_token)# convert tokens to DFM
dim(hotel_demo_dfm)
```

```
## [1]  3 32
```

```
hotel_demo_dfm[1:3,1:10]
```

```
## Document-feature matrix of: 3 documents, 10 features (63.33% sparse) and 0 docvars.
##        features
## docs    this residence marriott is not fancy it was clean good
##   text1    1         1        1  1   1     1  0   0     0    0
##   text2    0         0        0  0   0     0  1   1     1    1
##   text3    0         1        0  0   0     0  0   0     0    0
```

```
sparsity(hotel_demo_dfm)
```

```
## [1] 0.6458333
```

We are viewing 3 rows and 10 columns (unique words/tokens) as we abbreviate the full output. Even in a small text corpora (collection of documents) like this, there are so many zeros and so many columns (32 unique words represented in 32 columns). We have run into the curse of dimensionality and the issue of sparsity common in text data (approximately 65% data is sparse). Some of the ways to reduce the dimensions in preprocessing are through stemming, lemmatization etc. Later in the semester, we will look at more advanced techniques of data reduction.