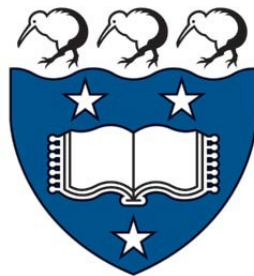


Text Analytics

Jason Peter Cairns

Supervised by Chris Wild



Bachelor of Science (Honours)
Department of Statistics
The University of Auckland
New Zealand

Todo list

should this be an abstract?	9
---------------------------------------	---

Acknowledgements

Contents

Listings	5
Tables	6
Figures	7
1 Introduction	9
1.1 Intention	9
1.2 Background: Text Analytics (incl. examples)	9
1.2.1 common functions: sentiment, summarisation, scoring	9
1.2.2 Existing Systems	10
1.3 Background: inZight	10
1.3.1 What inZight is - capabilities, popularity, etc.	10
1.3.2 how our program fits in - shiny, inzicht lite etc.	10
1.4 Literature Review (existing packages in R)	11
1.4.1 Copy over from notes, flesh out a bit	11
1.4.2 Praise tidytext book, complain about the package	11
1.5 Scope of work	11
2 Text Analytics Prolusion	12
2.1 overview	12
2.1.1 Explain broadness of term	12
2.1.2 compile glossary from terms here	12
2.1.3 Areas of text analytics in a data science framework	12
2.1.4 what we have done	12
2.1.5 what we haven't done	12
2.2 terms	12
2.2.1 terms and their centrality	14
2.2.2 generalisation: n-grams, sentences etc.	14
2.3 Historical Background	14
2.3.1 computer science vs statistics - reflection in data science	14
2.4 Processing	14
2.4.1 why process	14

2.4.2	stopwords, lemmatisation etc.	14
2.4.3	modelling vs db joins - more info in notes	14
2.5	scores & statistics	14
2.5.1	why compute scores & statistics	14
2.5.2	scoring - tf-idf, word count	14
2.5.3	Suggestions for further research - more on the statistics of words	14
2.5.4	recount the book of John text analysis	14
2.6	Sentiment	14
2.6.1	why sentiment	14
2.6.2	Process of sentiment	14
2.6.3	sentiment modelling vs db joins	14
2.6.4	our implementation and why	14
2.6.5	reviews	14
2.6.6	issues	14
2.7	Summarisation	14
2.7.1	why compute summarisations	14
2.7.2	lexrank, textrank - include notes on lexrank	14
2.7.3	other methods	14
2.7.4	reddit bot example	14
2.8	what we didn't do (yet)	14
2.8.1	topic modelling	14
2.8.2	Term correlation	14
2.8.3	modelling based on linguistic features	14
2.9	Visualisation	14
2.9.1	talk about score vs structure	14
2.9.2	complain about tag clouds	14
2.9.3	talk about ggpage	14
2.9.4	discuss our experimentations with some alternative visualisations	14
3	Program Structure & Development	15
3.0.1	why R	15
3.0.2	Why Shiny	15
3.0.3	why tidyverse	15
3.0.4	Git	15
3.0.5	possible future: datatables, futures, etc.	15
3.0.6	why functional	15
3.0.7	Why lossless data	15
3.1	Program Architecture	15
3.1.1	Why structure it like it has been	15
3.1.2	make graph of architecture	15
3.1.3	Describe package and package creation	15

3.1.4	following three sections copy and paste from the notes	
	- buffing up as necessary	15
3.1.5	include screenshots	15
3.2	Preparation	15
3.2.1	Importing	16
3.2.2	Formatting	18
3.2.3	Filtering	18
3.2.4	Lemmatisation	19
3.2.5	Stemming	19
3.3	Insight	19
3.4	Visualisation	19
3.5	User Interface	19
4	Conclusion	20
4.1	Summary	20
4.1.1	summarise successes	20
4.1.2	summarise failures	20
4.1.3	general thoughts on the topic	20
4.2	Recommendations	20
4.2.1	educational potential of text analytics	20
4.2.2	what else remains	20
5	Appendix	21
	Glossary	38
	Index	38
	Bibliography	39

List of source codes

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Intention

Text Analytics serves to glean insight from a body of text. Within the broad category of text analytics, we seek to answer questions about what the text is communicating, what is felt about it, and how this information is structured. In this dissertation, we demonstrate the creation of a user-friendly program to perform text analytics functions using modern R with the Shiny web application framework. In a literate style, we illustrate top-down the structure of such a program, as well as the data structures and computational processes that have established their value for such a program.

should
this be
an ab-
stract?

1.2 Background: Text Analytics (incl. examples)

1.2.1 common functions: sentiment, summarisation, scoring

Text Analytics is comprised of a variety of processes and techniques to extract information from text. The text almost always requires some initial processing. Some of the following functions have proven utility, and are expanded upon in chapter 2;

- **Sentiment:** In order to answer what emotions are conveyed in a text, sentiment analysis is commonly performed. The technique yields some measure of what is represented in an emotional sense by the text, with a range different methods and their associated outputs allowing for different forms of the analysis. Sentiment analysis won't pick up the subtle nuances that a human reader would, but generally gives reasonable output over the extent of a text.
- **Associated Words:** The meaning of a text is dependent on the structure between and within words. Looking at how words are associated,

through correlation, common sequences, visualisation of sections, etc., allow for a clear high-level assessment of the associations between words. The higher level not only saves individual efforts, but will demonstrate any emergent properties inherent to a text, in a way that a direct reading won't necessarily reveal.

- **Summarisation:** Automation of an executive summary, or a list of key words, typically falls under the purview of summarisation. The primary aim is to rank and select the most “representative” words or sentences from a text. A few major techniques dominate, being somewhat complex in nature. The results are generally surprisingly well representative of a text.
- **Feature Counts:** The simplest quantitative measure is very often the most informative; from simple word counts, to selective counts of sentences within groups, counting features can reveal how much written weighting is given to various elements, aiding insight into both structure and sentiment simultaneously.

1.2.2 Existing Systems

There are several existing systems in the field of Text Analytics. The field was initially nurtured as a sub-field of Computer Science, being computationally-dependent in nature. More recently, there has been increasing statistical interest. The existing systems reflect this; most older text analytics programs were Artificial Intelligence focussed, being experimental in nature, typically composed in lisp. More recently, major statistical programs have been incorporating text analytic features, with a few smaller text analytics specific programs appearing. SAS, SPSS, and R are all examples of major statistical processing systems, with recent additions of text analytics capabilities. An overview of R packages aiding in text analytics will be given in section 1.4.

1.3 Background: inZight

1.3.1 What inZight is - capabilities, popularity, etc.

1.3.2 how our program fits in - shiny, inZight lite etc.

Our program will form part of the suite of modules extending inZight. It provides a simple GUI interface to rapidly perform common text analyses. The primary audience are those learning the fundamentals of text analysis and statistics, which could include students of the traditional text analytics fields of Statistics and Computer Science, but can and should include students of Linguistics, Communications, Law, History, and any other text-based field. Beyond the educational aspects of the program, it is fully functional for actual use for general text analysis.

1.4 Literature Review (existing packages in R)

1.4.1 Copy over from notes, flesh out a bit

1.4.2 Praise tidytext book, complain about the package

1.5 Scope of work

The total scope possible for text analytics is enormous; our time in creating this program is not, thus it is essential that we limit the scope. There are two primary areas with which we created the limitations: Text type, and analysis type.

By limiting the forms of text we work with, we can spend less effort on consideration of every single possible import and transformation case, and more time on the actual design of analysis. The simplest means with which to create the limitation exists in allowing only import of particular text files — in this case, we allow for flat `.txt` files, as well as tabular `.csv` and `.xlsx` files. What we do not provide (though by design leaving open to the future possibility of including) is access in-program to common text sources through their API, such as Twitter or Project Gutenberg.

Through focussing on dictionary-based, rather than model-based analyses, we have avoided much of the associated complexities. An example of this is given in that it is common to categorise words based on their grammatical category, then use models that take this into account. By avoiding that (again, keeping the design flexible enough to allow for this in the future), we have been able to get far more functionality implemented in a shorter amount of time, with the analyses still performing soundly. Additionally, we keep the focus on the general audience, as it is typically more advanced, linguistically-trained users who would make intelligent use of such analyses.

Chapter 2

Text Analytics Prolusion

2.1 overview

Most importantly, words must be extracted, serving as the basic unit of analysis, from which more complex items may be derived.

2.1.1 Explain broadness of term

2.1.2 compile glossary from terms here

2.1.3 Areas of text analytics in a data science framework

2.1.4 what we have done

2.1.5 what we haven't done

2.2 terms

term

2.2.1 terms and their centrality

2.2.2 generalisation: n-grams, sentences etc.

2.3 Historical Background

2.3.1 computer science vs statistics - reflection in data science

2.4 Processing

2.4.1 why process

2.4.2 stopwords, lemmatisation etc.

2.4.3 modelling vs db joins - more info in notes

2.5 scores & statistics

2.5.1 why compute scores & statistics

2.5.2 scoring - tf-idf, word count

2.5.3 Suggestions for further research - more on the statistics of words

2.5.4 recount the book of John text analysis

2.6 Sentiment

2.6.1 why sentiment

2.6.2 Process of sentiment

2.6.3 sentiment modelling vs db joins

2.6.4 our implementation and why

2.6.5 reviews

2.6.6 issues

2.7 Summarisation

2.7.1 why compute summarisations

2.7.2 lexrank, textrank - include notes on lexrank

2.7.3 other methods

2.7.4 reddit bot example

2.8 what we didn't do (yet)

2.8.1 topic modelling

2.8.2 Term correlation

2.8.3 modelling based on linguistic features

2.9 Visualisation

Chapter 3

Program Structure & Development

3.0.1 why R

3.0.2 Why Shiny

3.0.3 why tidyverse

3.0.4 Git

3.0.5 possible future: datatables, futures, etc.

3.0.6 why functional

3.0.7 Why lossless data

3.1 Program Architecture

3.1.1 Why structure it like it has been

3.1.2 make graph of architecture

3.1.3 Describe package and package creation

3.1.4 following three sections copy and paste from the notes
- buffing up as necessary

3.1.5 include screenshots

3.2 Preparation

The first step in all text analysis is to import the text data and capture it into a data structure that reasonable analysis can be performed on.

3.2.1 Importing

Text must first be brought in from an outside source to be useful for the program. The import functions are such that all text from different files exist in dataframes of equivalent structure. The primary differences are that each row of an imported `.txt` file corresponds to a single line, whereas each row of an imported tabular file corresponds to the row of the tabular file. Importantly for tabular files, the column of the text intended for analysis must be given the header of “text” prior to import.

Import `.txt`

The following is the simple function used in the import of `.txt` files:

```
1  #' Import text file
2  #'
3  #' @param filepath a string indicating the relative or absolute
4  #'       filepath of the file to import
5  #'
6  #' @return a [tibble][tibble::tibble-package] of each row
7  #'       corresponding to a line of the text file, with the column named
8  #'       "text"
9  import_txt <- function(filepath){
10     readr::read_lines(filepath) %>%
11       tibble::tibble(text=.)
12 }
```

Import `.csv`

CSV is a plaintext tabular format, with columns typically delimited by commas, and rows by new lines. A particular point of difference in the importation of tabular data and regular plaintext is that the text of interest for the analysis should be (as per tidy principles) in one column, with the rest being additional information that can be used for grouping or filtering. Thus, additional user input is required, in the specification of which column is the text column of interest. The following function is effectively just a wrapper around `readr::read_csv()`

```
1  #' Import csv file
2  #'
3  #' @param filepath a string indicating the relative or absolute
4  #'       filepath of the file to import
5  #'
6  #' @return a [tibble][tibble::tibble-package] of each row
7  #'       corresponding to a line of the text file, with the column named
8  #'       "text"
9  import_csv <- function(filepath){
10     readr::read_csv(filepath)
11 }
```

Import Excel

Unfortunately, much data exists in the Microsoft Excel format, but this must be catered for. As tabular data, it is treated equivalently to csv, with a wrapper around `readr::read_excel()`

```
1 #' Import excel file
2 #'
3 #' @param filepath a string indicating the relative or absolute
4 #'   filepath of the file to import
5 #'
6 #' @return a [tibble][tibble::tibble-package] of each row corresponding to a line of the text
7 #'   file, with the column named "text"
8 import_excel <- function(filepath){
9   readxl::read_excel(filepath) ## %>%
10   ## table_textcol()
11 }
```

Import Wrapper

To have just one function required to import files, we define two functions; one that imports any file, and one making use of it to import multiple files.

The base wrapper function takes in the filename, and other relevant information, handling the importation process. It also stamps in the name of the document as a column.

```
1 #' Base case for file import
2 #'
3 #' @param filepath string filepath of file for import
4 #'
5 #' @return imported file with document id
6 import_base_file <- function(filepath){
7   filetype <- get_filetype(filepath)
8   filename <- basename(filepath)
9   if (filetype == "csv"){
10     imported <- import_csv(filepath)
11   } else if (filetype == "xlsx" | filetype == "xls") {
12     imported <- import_excel(filepath)
13   } else {
14     imported <- import_txt(filepath)
15   }
16   imported %>%
17     dplyr::mutate(doc_id = filename)
18 }
```

The base file import is generalised to multiple files with a multiple import function: this will be our sole import function

```
1 #' Import any number of files
2 #'
3 #' @param filepaths char vector of filepaths
4 #'
5 #' @return a [tibble][tibble::tibble-package] imported files with
6 #'   document id
```

```

7 #'
8 #' @export
9 import_files <- function(filepaths){
10   filepaths %>%
11     purrr::map(import_base_file) %>%
12     dplyr::bind_rows()
13 }

```

3.2.2 Formatting

From the imported files, we work at formatting them into a lossless and efficient data structure that any analysis can make use of. Our solution to this constraint is to separate and ID by each word in a dataframe. To do this, we take the line ID, the sentence ID, then the word ID, producing a dataframe that takes the following form:

line _{id}	sentence _{id}	word _{id}	word
1	1	1	the
1	1	2	quick
2	1	3	brown

The reason for the ID columns is the preservation of the structure of the text; If required, the original text can be reconstructed in entirety, sans minor punctuation differences. The following function automatically formats any data of the format returned by the initial import functions.

```

1 #' formats imported data into an analysis-ready format
2 #'
3 #' @param data a tibble formatted with a text and (optional) group
4 #'       column
5 #'
6 #' @return a [tibble][tibble::tibble-package] formatted such that columns correspond to
7 #'       identifiers of group, line, sentence, word (groups ignored)
8 #'
9 #' @export
10 format_data <- function(data){
11   data %>%
12     dplyr::mutate(line_id = dplyr::row_number()) %>%
13     tidytext::unnest_tokens(output = sentence, input = text, token = "sentences", to_lower = FALSE) %>%
14     dplyr::mutate(sentence_id = dplyr::row_number()) %>%
15     dplyr::group_by(sentence_id, add=TRUE) %>%
16     dplyr::group_modify(~ {
17       .x %>%
18         tidytext::unnest_tokens(output = word, input = sentence, token = "words", to_lower=FALSE) %>%
19         dplyr::mutate(word_id = dplyr::row_number())
20     }) %>%
21     ungroup_by("sentence_id")
22 }

```

3.2.3 Filtering

Filtering of text is implemented directly with the `dplyr::filter()` function, directly in the server of the shiny app. Filtering can take place multiple

times throughout an analysis. The program is flexible enough such that after some initial analytics have been done in the insight layer, preparation can be returned to and the text can be filtered on based on the analytics.

3.2.4 Lemmatisation

Lemmatisation is effectively the process of getting words into dictionary form. It is a very complex, stochastic procedure, as natural languages don't follow consistent and clear rules all the time. Hence, models have to be used. Despite the burden, it is generally worthwhile to lemmatise words for analytics, as there are many cases of words not being considered significant, purely due to taking so many different forms relative to others. Additionally, stopwords work better when considering just the lemmatised form, rather than attempting to exhaustively cover every possible form of a word. `textstem` is an R package allowing for easy lemmatisation, with its function `lemmatize_words()` transforming a vector of words into their lemmatised forms (thus being compatible with `mutate()` straight out of the box). We have the lemmatisation in this program managed completely by this single function in the server end of the shiny app. The package `Udpipe` was another option, but it requires downloading model files, and performs far more in depth linguistic determinations such as parts-of-speech tagging, that we don't need at this point. Worth noting is that, like stopwords, there are different dictionaries available for the lemmatisation process, but we will use the default, as testing has shown it to be the simplest to set up and just as reliable as the rest.

3.2.5 Stemming

Stemming is far simpler than lemmatisation, being the removal of word endings. This doesn't require as complex a model, as it is deterministic. It is not quite as effective, as the base word ending is not concatenated back on at the tail, so we are left with word stumps and morphemes. However, it may sometimes be useful when the lemmatisation model isn't working effectively, and `textstem` provides the capability with `stem_words()`. We have not implemented this yet, as it is not as essential to an analysis.

3.3 Insight

3.4 Visualisation

3.5 User Interface

Chapter 4

Conclusion

4.1 Summary

4.1.1 summarise successes

4.1.2 summarise failures

4.1.3 general thoughts on the topic

4.2 Recommendations

4.2.1 educational potential of text analytics

4.2.2 what else remains

Chapter 5

Appendix

The following pages are a copy of the documentation for the R package created as a part of this dissertation. They were automatically generated through the Roxygen2 system.

Package ‘inzightta’

August 16, 2019

Title iNZight Text Analytics

Version 0.0.0.9000

Description Provides text analytics functions for the importation, analysis, and visualisation of text. This package is designed specifically for output in shiny, with the analytical functions all working well with dplyr tools.

License GPL-3

Encoding UTF-8

LazyData true

Imports readr,
tibble,
stringr,
dplyr,
readxl,
purrr,
tidytext,
textstem,
magrittr,
stats,
textrank,
lexRankr

RoxygenNote 6.1.1

R topics documented:

aggregate_sentiment	2
determine_stopwords	3
format_data	3
get_bigram	4
get_chapters	4
get_filetype	5
get_parts	5
get_search	6

get_sections	6
get_sw	7
get_valid_input	7
ifexp	8
import_base_file	8
import_csv	9
import_excel	9
import_files	10
import_txt	10
index_bigram	11
keywords_tr	11
key_sentences	12
table_textcol	12
term_count	13
term_freq	13
text_prep	14
ungroup_by	14
word_sentiment	15
Index	16

aggregate_sentiment	<i>Get statistics for sentiment over some group, such as sentence.</i>
---------------------	--

Description

Get statistics for sentiment over some group, such as sentence.

Usage

```
aggregate_sentiment(.data, aggregate_on, statistic)
```

Arguments

.data	character vector of words
aggregate_on	vector to aggregate .data over; ideally, sentence_id, but could be chapter, document, etc.
statistic	function that accepts na.rm argument; e.g. mean, median, sd.

determine_stopwords	<i>determine stopword status</i>
---------------------	----------------------------------

Description

determine stopword status

Usage

```
determine_stopwords(.data, ...)
```

Arguments

.data	vector of words
...	arguments of get_sw

Value

a [tibble][tibble::tibble-package] equivalent to the input dataframe, with an additional stopword column

format_data	<i>formats imported data into an analysis-ready format</i>
-------------	--

Description

formats imported data into an analysis-ready format

Usage

```
format_data(data)
```

Arguments

data	a tibble formatted with a text and (optional) group column
------	--

Value

a [tibble][tibble::tibble-package] formatted such that columns correspond to identifiers of group, line, sentence, word (groups ignored)

get_bigram	<i>Determine bigrams</i>
------------	--------------------------

Description

Determine bigrams

Usage

```
get_bigram(.data)
```

Arguments

.data character vector of words

Value

character vector of bigrams

get_chapters	<i>sections text based on chapters</i>
--------------	--

Description

sections text based on chapters

Usage

```
get_chapters(.data)
```

Arguments

.data vector to section

Value

vector of same length as .data with chapter numbers

get_filetype	<i>Get filetype</i>
--------------	---------------------

Description

Get filetype

Usage

```
get_filetype(filepath)
```

Arguments

filepath	string filepath of document
----------	-----------------------------

Value

filetype (string) - NA if no extension

get_parts	<i>sections text based on parts</i>
-----------	-------------------------------------

Description

sections text based on parts

Usage

```
get_parts(.data)
```

Arguments

.data	vector to section
-------	-------------------

Value

vector of same length as .data with part numbers

get_search	<i>creates a search closure to section text</i>
------------	---

Description

creates a search closure to section text

Usage

```
get_search(search)
```

Arguments

search	a string regexp for the term to seperate on, e.g. "Chapter"
--------	---

Value

closure over search expression

get_sections	<i>sections text based on sections</i>
--------------	--

Description

sections text based on sections

Usage

```
get_sections(.data)
```

Arguments

.data	vector to section
-------	-------------------

Value

vector of same length as .data with section numbers

get_sw	<i>Gets stopwords from a default list and user-provided list</i>
--------	--

Description

Gets stopwords from a default list and user-provided list

Usage

```
get_sw(lexicon = "snowball", addl = NA)
```

Arguments

lexicon	a string name of a stopwords list, one of "smart", "snowball", or "onix"
addl	user defined character vector of additional stopwords, each element being a stop-word

Value

a [tibble][tibble::tibble-package] with one column named "word"

get_valid_input	<i>helper function to get valid input (recursively)</i>
-----------------	---

Description

helper function to get valid input (recursively)

Usage

```
get_valid_input(options, init = TRUE)
```

Arguments

options	vector of options that valid input should be drawn from
init	whether this is the initial attempt, used only as recursive information

Value

readline output that exists in the vector of options

ifexp	<i>scheme-like if expression, without restriction of returning same-size table of .test, as ifelse() does</i>
-------	---

Description

scheme-like if expression, without restriction of returning same-size table of .test, as ifelse() does

Usage

```
ifexp(.test, true, false)
```

Arguments

.test	predicate to test
true	expression to return if .test evals to TRUE
false	expression to return if .test evals to TRUE

Value

either true or false

import_base_file	<i>Base case for file import</i>
------------------	----------------------------------

Description

Base case for file import

Usage

```
import_base_file(filepath)
```

Arguments

filepath	string filepath of file for import
----------	------------------------------------

Value

imported file with document id

import_csv	<i>Import csv file</i>
------------	------------------------

Description

Import csv file

Usage

```
import_csv(filepath)
```

Arguments

filepath a string indicating the relative or absolute filepath of the file to import

Value

a [tibble][tibble::tibble-package] of each row corresponding to a line of the text file, with the column named "text"

import_excel	<i>Import excel file</i>
--------------	--------------------------

Description

Import excel file

Usage

```
import_excel(filepath)
```

Arguments

filepath a string indicating the relative or absolute filepath of the file to import

Value

a [tibble][tibble::tibble-package] of each row corresponding to a line of the text file, with the column named "text"

import_files	<i>Import any number of files</i>
--------------	-----------------------------------

Description

Import any number of files

Usage

```
import_files(filepaths)
```

Arguments

filepaths	char vector of filepaths
-----------	--------------------------

Value

a [tibble][tibble::tibble-package] imported files with document id

import_txt	<i>Import text file</i>
------------	-------------------------

Description

Import text file

Usage

```
import_txt(filepath)
```

Arguments

filepath	a string indicating the relative or absolute filepath of the file to import
----------	---

Value

a [tibble][tibble::tibble-package] of each row corresponding to a line of the text file, with the column named "text"

index_bigram	<i>get bigram at index i of list1 & 2</i>
--------------	---

Description

get bigram at index i of list1 & 2

Usage

```
index_bigram(i, list1, list2)
```

Arguments

i	numeric index to attain bigram at
list1	list or vector for first bigram token
list2	list or vector for second bigram token

Value

bigram of list1 and list2 at index i, skipping NA's

keywords_tr	<i>Determine textrank score for vector of words</i>
-------------	---

Description

Determine textrank score for vector of words

Usage

```
keywords_tr(.data)
```

Arguments

.data	character vector of words
-------	---------------------------

Value

vector of scores for each word

key_sentences	<i>get score for key sentences as per Lexrank</i>
---------------	---

Description

get score for key sentences as per Lexrank

Usage

```
key_sentences(.data, aggregate_on)
```

Arguments

.data	character vector of words
aggregate_on	vector to aggregate .data over; ideally, sentence_id

table_textcol	<i>Interactively determine and automatically mark the text column of a table</i>
---------------	--

Description

Interactively determine and automatically mark the text column of a table

Usage

```
table_textcol(data)
```

Arguments

data	dataframe with column requiring marking
------	---

Value

same dataframe with text column renamed to "text"

term_count	<i>Determine the number of terms at each aggregate level</i>
------------	--

Description

Determine the number of terms at each aggregate level

Usage

```
term_count(.data, aggregate_on)
```

Arguments

.data	character vector of terms
aggregate_on	vector to split .data on for insight

Value

vector of number of terms for each aggregate level, same length as .data

term_freq	<i>Determine term frequency</i>
-----------	---------------------------------

Description

Determine term frequency

Usage

```
term_freq(.data)
```

Arguments

.data	character vector of terms
-------	---------------------------

Value

numeric vector of term frequencies

text_prep	<i>takes imported one-line-per-row data and prepares it for later analysis</i>
-----------	--

Description

takes imported one-line-per-row data and prepares it for later analysis

Usage

```
text_prep(.data, lemmatize = TRUE, stopwords = TRUE,
  sw_lexicon = "snowball", addl_stopwords = NA)
```

Arguments

.data	tibble with one line of text per row
lemmatize	boolean, whether to lemmatize or not
stopwords	boolean, whether to remove stopwords or not
sw_lexicon	string, lexicon with which to remove stopwords
addl_stopwords	char vector of user-supplied stopwords

Value

a [tibble][tibble::tibble-package] with one token per line, stopwords removed leaving NA values, column for analysis named "text"

ungroup_by	<i>helper function to ungroup for dplyr. functions equivalently to group_by() but with standard (string) evaluation</i>
------------	---

Description

helper function to ungroup for dplyr. functions equivalently to group_by() but with standard (string) evaluation

Usage

```
ungroup_by(x, ...)
```

Arguments

x	tibble to perform function on
...	string of groups to ungroup on

Value

x with ... no longer grouped upon

word_sentiment	<i>Determine sentiment of words</i>
----------------	-------------------------------------

Description

Determine sentiment of words

Usage

```
word_sentiment(.data, lexicon = "afinn")
```

Arguments

.data	vector of words
lexicon	sentiment lexicon to use, based on the corpus provided by tidytext

Value

vector with sentiment score of each word in the vector

Index

`aggregate_sentiment`, [2](#)

`determine_stopwords`, [3](#)

`format_data`, [3](#)

`get_bigram`, [4](#)

`get_chapters`, [4](#)

`get_filetype`, [5](#)

`get_parts`, [5](#)

`get_search`, [6](#)

`get_sections`, [6](#)

`get_sw`, [7](#)

`get_valid_input`, [7](#)

`ifexp`, [8](#)

`import_base_file`, [8](#)

`import_csv`, [9](#)

`import_excel`, [9](#)

`import_files`, [10](#)

`import_txt`, [10](#)

`index_bigram`, [11](#)

`key_sentences`, [12](#)

`keywords_tr`, [11](#)

`table_textcol`, [12](#)

`term_count`, [13](#)

`term_freq`, [13](#)

`text_prep`, [14](#)

`ungroup_by`, [14](#)

`word_sentiment`, [15](#)

Glossary

term “a word or expression that has a precise meaning in some uses or is peculiar to a science, art, profession, or subject”[1] — here text analysts have capitalised on the generalisation of “term” to include subcomponents or aggregations of words. 9

Bibliography

- [1] Merriam-Webster Dictionary, ed. *Term — Definition of Term*. 17th Aug. 2019. URL: <https://www.merriam-webster.com/dictionary/term> (cit. on p. 38).