Projects for Text Version Comparison and Analytics in R

Contents

1 Overview

Status

R code: 1625 *C++ code:* 112 *test code:* 1010

Version

0.1.12

Description

Provides data structures and methods for manual as wells as automated R based text comparison and text as well as change coding.

Funding

This software was created as part of the "Institutional Design in Western European Democracies" research project, funded by DFG (Deutsche Forschungsgemeinschaft), lead by Ulrich Sieberer and based at University Konstanz.

License

MIT + file LICENSE Peter Meissner retep.meissner@gmail.com [aut, cre] Ulrich Sieberer ulrich.sieberer@uni-bamberg.de [cph] University of Konstanz willkommen@uni-konstanz.de [cph]

Citation

Meißner P (2016). *diffrprojects: Projects for Text Version Comparison and Analytics in R*. R package version 0.1.12, <URL: https://github.com/petermeissner/diffrprojects>.

Sieberer U, Meißner P, Keh J and Müller W (2016). "Mapping and Explaining Parliamentary Rule Changes in Europe: A Research Program." *Legislative Studies Quarterly*, *41*(1), pp. 61-88. ISSN 1939-9162, doi: 10.1111/lsq.12106 (URL: http://doi.org/10.1111/lsq.12106), <URL: http://dx.doi.org/10.1111/lsq.12106>.

BibTex for citing

```
toBibtex(citation("diffrprojects"))
```

Installation

stable CRAN version

```
install.packages("diffrprojects")
library(rtext)
```

(stable) development version

```
standard_repos <- options("repos")$repos
install.packages(
  "diffrprojects",
  repos = c(standard_repos, "https://petermeissner.github.io/drat/")
)
library(rtext)
```

Contribution

Note, that this package uses a Contributor Code of Conduct. By participating in this project you agree to abide by its terms: http://contributor-covenant.org/version/1/0/0/ (basically this should be a place where people get along with each other respectfully and nicely, because it's simply more fun that way for everybody)

Contributions are very much welcome, e.g. in the form of:

- **typo fixing** (edit file directly on Github)
- **bug reporting** (file an issue - after having searched if the issue came up before - as - if possible - minimal reproducable example)
- **extending help files** (e.g. edit the respective files directly on Github or fork the package and later on make a pull request; note, that the package use roxygen2 for easing documentation)
- **writing example** (e.g. edit the respective files directly on Github or fork the package and later on make a pull request; note, that the package use roxygen2 for easing documentation)
- **vignette writing** (file an issue first so that we can discuss htings than fork the package and later on make a pull request)
- **test writing** (have a look at the test coverage than fork the package and later on make a pull request)
- **feature suggestions** (file an issue describing the idea, why this is important, possible alternative solutions and an example)
- **general discussion** of approach and or implementation (file an issue)
- implementation **improvements** (file an issue naming whats to be improved, why and how)

2 Usage

2.1 Fast Introduction for the Impatient

For those in a hurry here is a very brief

```
# loading package
library(diffrprojects)


# the first chapter of Robinson Crusoe from three different sources
rcs <- rtext:::testfile(pattern="rc.*ch1.txt", full.names = TRUE)


# creating a new project
dp <- diffrproject$new()


# setting options
dp$options$verbose <- FALSE


# adding texts to the corpus
dp$text_add(text_file = rcs)
dp$text_data(1) %>% head(11)
```

```
##     i char         name
## 1   1    T rc_1_ch1.txt
## 2   2    h rc_1_ch1.txt
## 3   3    e rc_1_ch1.txt
## 4   4      rc_1_ch1.txt
## 5   5    P rc_1_ch1.txt
## 6   6    r rc_1_ch1.txt
## 7   7    o rc_1_ch1.txt
## 8   8    j rc_1_ch1.txt
## 9   9    e rc_1_ch1.txt
## 10 10    c rc_1_ch1.txt
## 11 11    t rc_1_ch1.txt
```

```
# linking the files (which file should be compared to which)
dp$text_link()
dp$link %>% as.data.frame()
```

```
##           from           to                      link
## 1 rc_1_ch1.txt rc_2_ch1.txt rc_1_ch1.txt~rc_2_ch1.txt
## 2 rc_2_ch1.txt rc_3_ch1.txt rc_2_ch1.txt~rc_3_ch1.txt
```

```
# calculating text alignments
dp$text_align(tokenizer=text_tokenize_words)
dp$alignment[[1]] %>% head(30)
```

```
##    alignment_i token_i_1 token_i_2 distance      type from_1 to_1 from_2  to_2
## 1            1         1      1932        0 no-change      1    3  10326 10328
## 2            2         2        NA        7  deletion      5   11     NA    NA
## 3            3         3        NA        9  deletion     13   21     NA    NA
## 4            4         4        NA        5  deletion     23   27     NA    NA
```

```
## 5            5         5      1932        0 no-change     30   32  10326 10328
## 6            6         6        NA        4  deletion     34   37     NA    NA
## 7            7         7        87        0 no-change     39   41    513   515
## 8            8         8        NA       10  deletion     43   52     NA    NA
## 9            9         9        57        0 no-change     54   55    355   356
## 10          10        10         3        0 no-change     57   64     15    22
## 11          11        11         4        0 no-change     66   71     24    29
## 12          12        12        85        0 no-change     74   75    497   498
## 13          13        13         1        0 no-change     77   82      1     6
## 14          14        14         2        0 no-change     84   88      8    12
## 15          15        15      1520        0 no-change     92   95   8187  8190
## 16          16        16        NA        5  deletion     97  101     NA    NA
## 17          17        17      3667        0 no-change    103  104  19215 19216
## 18          18        18       263        0 no-change    106  108   1495  1497
## 19          19        19        51        0 no-change    110  112    328   330
## 20          20        20        NA        3  deletion    114  116     NA    NA
## 21          21        21        57        0 no-change    118  119    355   356
## 22          22        22        NA        6  deletion    121  126     NA    NA
## 23          23        23        NA        8  deletion    128  135     NA    NA
## 24          24        24        50        0 no-change    137  138    325   326
## 25          25        25        51        0 no-change    140  142    328   330
## 26          26        26        NA        6  deletion    144  149     NA    NA
## 27          27        27        NA        6  deletion    151  156     NA    NA
## 28          28        28        87        0 no-change    158  160    513   515
## 29          29        29       513        0 no-change    162  165   2853  2856
## 30          30        30       306        0 no-change    167  171   1724  1728
```

2.2 Creating a Diffrprojects Instance

To create a diffrproject we use the diffrproject creator object - it´s simply an object with a function that knows how to create a project.

Creating a project looks like this:

```
library(diffrprojects)
```

```
## Warning: package 'rtext' was built under R version 3.3.2
```

```
dp <- diffrproject$new()
```

Et violà - we created a first, for now empty, project that we will use throughout the tutorial.

2.3 Some Help Please

To get a better idea about what this thing called *diffrproject* really is, you can consult its help page which gives a broad overview over its capabilities:

```
?diffrproject
```

Another way is to call the ls() method. This will present us with a data frame listing, all fields where data is stored and all the methods (aka object specific functions) of our diffrprojects instance. Those methods and fields located in *private* are not for the user to mess around with while non-private (*self* aka public) data fields can be read by the user and public methods can be triggered by the user to manipulate the data or retrieve data in a specific format.

```
dp$ls()
```

```
##                         name   where                     class
## 1               execute_load private                  function
## 2                       hash private                  function
## 3                     hashed private                  function
## 5               prepare_save private                  function
## 4                     hashes private                      list
## 9             alignment_data    self alignment_data_list, list
## 6                  alignment    self      alignment_list, list
## 21                      link    self      alignment_list, list
## 7               alignment_add    self                  function
## 8             alignment_code    self                  function
## 10       alignment_data_full    self                  function
## 11        alignment_data_set    self                  function
## 12          alignment_delete    self                  function
## 13                     clone    self                  function
## 14                     debug    self                  function
## 15                export_csv    self                  function
## 16              export_sqlite    self                  function
## 17                       get    self                  function
## 18                import_csv    self                  function
## 19              import_sqlite    self                  function
## 20                 initialize    self                  function
## 22                       load    self                  function
## 23                         ls    self                  function
## 24                    message    self                  function
## 27                       save    self                  function
## 29                   text_add    self                  function
## 30                 text_align    self                  function
## 31                  text_code    self                  function
## 32 text_code_alignment_token    self                  function
## 33           text_code_regex    self                  function
## 34                 text_data    self                  function
## 35         text_data_inherit    self                  function
## 36               text_delete    self                  function
## 37                 text_link    self                  function
## 38            text_meta_data    self                  function
## 39  tokenize_text_data_lines    self                  function
## 40  tokenize_text_data_regex    self                  function
## 41  tokenize_text_data_words    self                  function
## 42                   warning    self                  function
## 25                      meta    self                      list
## 26                   options    self                      list
## 28                      text    self                      list
```

The base R class() function furthermore reveals from which classes the diffrproject class inherits:

```
class(dp)
```

```
## [1] "diffrproject"      "dp_inherit"        "dp_align"          "dp_export"
## [5] "rtext_loadsave"    "dp_base"           "R6_rtext_extended" "R6"
```

2.4 Adding Texts to Projects

Our diffrproject (`dp`) has one method called `text_add()` that allows to add texts to the project. Basically the method can be used in three different flavors: adding character vectors, adding texts stored on disk, or by adding rtext objects (see rtext package: https://CRAN.R-project.org/package=rtext; rtext objects are the way individual texts are represented within diffrprojects). For each of these used cases there is one option: `text`, `text_file`, `rtext`; respectively.

Below are shown examples using each of these methods:

adding text files

```
test_file1 <- stringb:::test_file("rc_1_ch1.txt")
test_file2 <- stringb:::test_file("rc_2_ch1.txt")
dp$text_add(text_file = c(test_file1, test_file2) )
```

adding rtext objects

```
test_file <- stringb:::test_file("rc_1_ch1.txt")
rt <- rtext$new( text_file = test_file)
dp$text_add(rtext = rt)
```

adding character vectors

```
test_file1 <- stringb:::test_file("rc_1_ch1.txt")
test_file2 <- stringb:::test_file("rc_2_ch1.txt")
cv <- ""
cv[1] <- text_read(test_file1, NULL)
cv[2] <- text_read(test_file2, NULL)
dp$text_add(text = cv)
```

In the last case make sure to put each text in one separate line. Functions like readLines() or text_read() read in texts such that each line corresponds to one element in a character vector. With e.g. text_read()'s tokenize parameter to NULL the text will be read in as one long string.

2.5 Piping Methods

Now is a good time to mention a feature of diffrprojects that comes in handy: All functions that do not explicitly extract data (those usually have some 'get' as part of their name) do return the object itself so that one can pipe together a series of method calls.

Consider the following example where we initiate a new diffrprojects instance and add two texts in just one pipe:

```
dp <-
  diffrproject$
  new()$
  text_add(text_version_1, name = "version1")$
  text_add(text_version_2, name = "version2")

length(dp$text)
```

```
## [1] 2
```

2.6 Getting Infos About Texts

If we want to get some general overview about the texts gathered in our project, we can use the text_meta_data() method to do so. The method has no parameters and returns a data.frame with several variables informing us about its source, length, encoding used for storage, and its name.

```
dp$text_meta_data()
```

```
##   text_file character encoding sourcetype     name
## 1      <NA>       479    UTF-8       text version1
## 2      <NA>       539    UTF-8       text version2
```

2.7 Showing Text

If you want to have a look at your texts you may do so by using the text's own text_show methods. Per default this method only shows the first 500 characters, but it can be set to higher numbers as well.

```
dp$text$version1$text_show(length=1000)
```

```
## This part of the
## document has stayed the
## same from version to
## version.  It shouldn't
## be shown if it doesn't
## change.  Otherwise, that
## would not be helping to
## compress the size of the
## changes.
##
## This paragraph contains
## text that is outdated.
## It will be deleted in the
## near future.
##
## It is important to spell
## check this dokument. On
## the other hand, a
## misspelled word isn't
## the end of the world.
## Nothing in the rest of
## this paragraph needs to
## be changed. Things can
## be added after it.
##
```

```
dp$text$version2$text_show(length=1000)
```

```
## This is an important
## notice! It should
## therefore be located at
## the beginning of this
## document!
##
## This part of the
## document has stayed the
```

```
## same from version to
## version.  It shouldn't
## be shown if it doesn't
## change.  Otherwise, that
## would not be helping to
## compress anything.
##
## It is important to spell
## check this document. On
## the other hand, a
## misspelled word isn't
## the end of the world.
## Nothing in the rest of
## this paragraph needs to
## be changed. Things can
## be added after it.
##
## This paragraph contains
## important new additions
## to this document.
```

2.8 Getting And Setting Infos About the Project

Similar to the text_meta_data() method we can access the projects meta data via data fields meta and options. But contrary to the text_meta_data() method that gathers data from all the texts within the project and does not allow for manipulation of the data, the data fields allow reading and writing.

First let us have a look and thereafter turn off the message notification service:

getting data fields

```
dp$options
```

```
## $verbose
## [1] TRUE
##
## $warning
## [1] TRUE
##
## $ask
## [1] TRUE
```

setting data fields

```
dp$options$verbose <- FALSE
```

(note, ask is deprecated and only remains for compatibility reasons but has no function anymore)

Now it´s time to have a look at the projects meta data. It tells us when the project was created, which path to use for SQLite exports, which path to use for saving data as in RData format and what is the projects id. The id is a hash of a time stamp as well as session information which should ensure uniqueness across space and time.

All these values can be manipulated by the user to her liking.

```
dp$meta
```

```
## $ts_created
```

```
## [1] "2016-11-06 10:21:03 UTC"
##
## $db_path
## [1] "./diffrproject.db"
##
## $file_path
## [1] ""
##
## $project_id
## [1] "33569831dacb2c5656dbe93f7794866a"
```

```
dp$meta$file_path = "./diffrproject.RData"
```

2.9 Deleting Texts

Of cause we can not only add texts but delete them from the project as well. For this purpose there is the text_delete() method.

Let's just add two texts and delete one by providing its index number and the second by providing its name to the text_delete() method.

```
dp$text_add(text = "nonesense", "n1")
dp$text_add(text = "nonesense", "n2")

dp$text_delete(3)
dp$text_delete("n2")

length(dp$text)
```

```
## [1] 2
```

```
names(dp$text)
```

```
## [1] "version1" "version2"
```

2.10 Defining Relationships Between Texts: Linking

The purpose of diffrprojects is to enable data collection on the difference of texts. Having filled a project with various texts, there are endless possibilities to form pairs of text for comparison and change measurement - where endless actually is equal to: $n^2 - n$.

Linking can be done via the text_link method which accepts either index numbers or text names for its from and to arguments (a third argument delete will delete a specified link if set to TRUE).

```
dp$text_link(from = 1, to = 2)
dp$text_link(from = 1, to = 2, delete = TRUE)
```

If no arguments are specified, text_link will link the first text to the second, the third to the fourth, the fourth to the fifths and so on.

```
dp$text_link()
```

To get an idea of what links are currently specified, we can directly access the link data field or/and ask R to transform the list found there into a data.frame.

```
dp$link
```

```
## $`version1~version2`
## $`version1~version2`$from
## [1] "version1"
##
## $`version1~version2`$to
## [1] "version2"
##
##
## attr(,"class")
## [1] "alignment_list" "list"
```

```
dp$link %>% as.data.frame()
```

```
##       from       to              link
## 1 version1 version2 version1~version2
```

2.11 Aligning Texts and Measuring Change

At the heart of each diffrproject lies the text_align method. This method compares two texts and tries to align parts of one text with parts of the other text. The first two arguments (`t1` and `t2`) are for specifying which pair of texts to compare - if left as-is, all text pairs that are specified within the link data field will be aligned.

Text parts are arbitrary character spans defined by the `tokenizer` argument. This argument expects a function splitting text into a token data.frame. If the tokenizer argument is left as-is, it will default to text_tokenize_lines function from the stringb package.

Text tokens can be pre-processed before alignment. The `clean` argument allows to hand over a function tranforming a charactr vector of text tokens into their clean counterparts.

The `ignore` arguments expects a function that is able to transform a character vector of tokens into a logical vector of same length, indicating which tokens to ignore throughout the alignment process and which to consider.

The next argument - `distance` - specifies which distance metrics to use to calculate distances between strings.

Since the text_align method basically is a wrapper around diff_align you can get more information via `?diff_align` and since again diff_align is a wrapper around stringdist from the stringdist package `?stringdist::stringdist` and also `?stringdist::‘stringdist-metrics‘` will provide further insights about possible metrics and how to use the rest of the arguments to text_align (these are passed through to stringdist).

Let's have an example using the Levenshtein distance to calculate distances between tokens (lines per default). Furthermore we allow the distance between two aligned tokens to be as large as 15. Tokens which do not find a partner below that distance are considered to have been deleted or respectively inserted. Tokens which find a partner with a non-zero distance which is not above the threshhold are considered changes - transformations of one token into the other.

The following shows the resulting list of alignment data.frames.

```
dp$text_align(distance = "lv", maxDist = 15)

dp$alignment
```

```
## $`version1~version2`
##    alignment_i token_i_1 token_i_2 distance      type from_1 to_1 from_2 to_2
## 1            1         1         6        0 no-change      1   16     97  112
## 2            2         2         7        0 no-change     18   40    114  136
```

```
## 3            3         3         8         0 no-change     42   61    138  157
## 4            4         4         9         0 no-change     63   84    159  180
## 5            5         5        10         0 no-change     86  107    182  203
## 6            6         6        11         0 no-change    109  132    205  228
## 7            7         7        12         0 no-change    134  156    230  252
## 8            8         8        13        14    change    158  181    254  271
## 9            9         9         5         8    change    183  190     86   94
## 11          10        10        23         0 no-change    193  215    475  497
## 12          11        11        25        13    change    217  238    523  539
## 13          12        12        NA        25  deletion    240  264     NA   NA
## 14          13        13         5        11    change    266  277     86   94
## 16          14        14        14         0 no-change    280  303    274  297
## 17          15        15        15         1    change    305  327    299  321
## 18          16        16        16         0 no-change    329  345    323  339
## 19          17        17        17         0 no-change    347  367    341  361
## 20          18        18        18         0 no-change    369  389    363  383
## 21          19        19        19         0 no-change    391  412    385  406
## 22          20        20        20         0 no-change    414  436    408  430
## 23          21        21        21         0 no-change    438  459    432  453
## 24          22        22        22         0 no-change    461  478    455  472
## 15          23        NA         1        20 insertion     NA   NA      1   20
## 25          24        NA         2        17 insertion     NA   NA     22   38
## 31          25        NA         3        23 insertion     NA   NA     40   62
## 41          26        NA         4        21 insertion     NA   NA     64   84
## 27          27        NA        24        23 insertion     NA   NA    499  521
##
## attr(,"class")
## [1] "alignment_list" "list"
```

To measure the change between those two texts we can e.g. aggregate the the distances by change type:

```
sum_up_changes <- function(x){
  x %>%
    dplyr::group_by(type) %>%
    dplyr::summarise(sum_of_change = sum(distance))
}

lapply( dp$alignment, sum_up_changes)
```

```
## $`version1~version2`
## # A tibble: 4 × 2
##        type sum_of_change
##       <chr>         <dbl>
## 1    change            47
## 2  deletion            25
## 3 insertion           104
## 4 no-change             0
```

2.12 Coding Texts

Now let us put some data into our diffrproject.

The most basic method to do so is simply called text_code. Text_code takes up to five arguments (the first three are mandatory), where one specifies the text to be coded (`text`, either by index number or by name), how the variable to store the information is called (`x`), and the index number or a vector of those indicating

which characters of the text should be coded. The last two parameters are optional and specify which value the variable should hold (`val`) and at which hierarchy level the coding is placed (`hl`, higher or equal hierarchy levels will overwrite existing codings of lower hierarchy level for the same text, character span, and variable).

```
dp$text_code(text = 1, x = "start", i=1:5, val = TRUE, hl = 0)
dp$text_code(text = "version2", x = "start", i=1:5, val = TRUE, hl = 0)
```

The text_code method is quite verbose and in most cases more suited to be accessed by a machine or algorithm than by a human. Therefore, there are three other methods to code text: text_code_regex, text_code_alignment_token, text_code_alignment_token_regex.

The text_code_regex method allows to search for text patterns and code a whole pattern instead of assigning codes character by character - the `i` argument of text_code gets replaced by a `pattern` argument. The in addition further arguments can be passed to the pattern search functions via `...` - see e.g. `?grep` for possible further arguments and https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html for a description of regular expressions in R.

In this example we are searching for the word *"it"* in text 1 and code each instance.

```
dp$text_code_regex(text = 1, x = "it", pattern = "\\bit\\b", ignore.case=TRUE)
```

Another variant of coding text is by using alignment tokens. Having alignment data availible, this allows for selecting: link, alignment and text while the other arguments from above stay the same.

```
# having a look at alignment number 4
dp$alignment[[1]][4,]
```

```
##   alignment_i token_i_1 token_i_2 distance      type from_1 to_1 from_2 to_2
## 4           4         4         9        0 no-change     63   84    159  180
```

```
# coding text connected by alignment number 4
dp$text_code_alignment_token(
  link        = 1,
  alignment_i = 4,
  text1       = TRUE,
  text2       = TRUE,
  x   = "token_coding",
  val = 4,
  hl  = 0
)
```

2.13 Getting Text Codings

The most basic way to get text data is to use the text_data method. This method will go through all or only selected texts, gather all the data stored there and put it into a neat data.frame where name identifies the text from which the data comes per name, char informs us about the character that was coded, and i refers to the characters position within the text. All other variables hold the data we added during the examples above.

```
dp$text_data(text = 1) %>% head()
```

```
##    i char start it token_coding     name
## 1  1    T  TRUE NA           NA version1
## 2  2    h  TRUE NA           NA version1
## 3  3    i  TRUE NA           NA version1
## 4  4    s  TRUE NA           NA version1
## 5  5       TRUE NA           NA version1
## 6 63    v    NA NA            4 version1
```

2.14 Aggregating Text Codings

The usage of text_data has its merits but often one is more interested in text data aggregated to a specific level. The following three aggregation functions offer a solution to this problem: tokenize_text_data_lines, tokenize_text_data_words, and tokenize_text_data_regex. These three methods make use of the similiary named methods provided by the rtext package.

One important thing to keep in mind is that using these methods implies aggregating several data values on character level into one data value at token level. Therefore there has to be some aggregation function to be involved. The default is to use the value that occurs most often on character level, if more than one distinct values occur more than once the first is choosen.

The aggregation function can be changed to whatever function the user seems appropriate by passing it to `aggregate_function` - as long as it reduces a vector of values into a vector with only one value.

The `join` argument allows to decide how text and data are joined into the resulting data.frame - left: all token, right: all data, full: token with or without data and data with or without token.

```
dp$tokenize_text_data_lines(
  text = 1,
  join = "right",
  aggregate_function =
    function(x){
      paste(x[1:3], collapse = ",")
    }
)
```

```
##   token_i from  to                     token is_token          start       it token_coding     name
## 1       1    1  16          This part of the     TRUE TRUE,TRUE,TRUE NA,NA,NA     NA,NA,NA version1
## 2       4   63  84    version.  It shouldn't     TRUE       NA,NA,NA NA,NA,NA        4,4,4 version1
## 3       5   86 107    be shown if it doesn't     TRUE       NA,NA,NA NA,NA,NA     NA,NA,NA version1
## 4      12  240 264 It will be deleted in the     TRUE       NA,NA,NA NA,NA,NA     NA,NA,NA version1
## 5      14  280 303  It is important to spell     TRUE       NA,NA,NA NA,NA,NA     NA,NA,NA version1
## 6      22  461 478        be added after it.     TRUE       NA,NA,NA NA,NA,NA     NA,NA,NA version1
```

2.15 Text Coding Inheritence

Having aligned two texts via token pairs another functionality of diffrprojects becomes availible: text coding inheritance via no-change tokens. This means that text codings can get copied to those tokens they are aligned with, given that they are considered the same - i.e. the distance equals zero and the change type therefore is no-change.

To show this feature we use the text_inherit method and we will start with a fresh example. A new project with two texts. The first text gets some codings, then they are aligned, and in a last step codings are transfered from one text to the other via the text_data_inherit method.

```
dp <-
  diffrproject$new()$
  text_add(text_version_1)$
  text_add(text_version_2)$
  text_code_regex(
    text    = 1,
    x       = "test1",
    pattern = "This part.*?change",
    val = "inherited"
  )$
```

```
  text_code_regex(
    text    = 1,
    x       = "test2",
    pattern = "This part.*?change",
    val = "inherited"
  )

dp$tokenize_text_data_lines(1)
```

```
##    token_i from  to                      token is_token     test1     test2     name
## 1        1    1  16          This part of the     TRUE inherited inherited noname_1
## 2        2   18  40   document has stayed the     TRUE inherited inherited noname_1
## 3        3   42  61      same from version to     TRUE inherited inherited noname_1
## 4        4   63  84    version.  It shouldn't     TRUE inherited inherited noname_1
## 5        5   86 107    be shown if it doesn't     TRUE inherited inherited noname_1
## 6        6  109 132  change.  Otherwise, that     TRUE inherited inherited noname_1
## 7        7  134 156   would not be helping to     TRUE      <NA>      <NA> noname_1
## 8        8  158 181  compress the size of the     TRUE      <NA>      <NA> noname_1
## 9        9  183 190                  changes.     TRUE      <NA>      <NA> noname_1
## 10      10  193 215   This paragraph contains     TRUE      <NA>      <NA> noname_1
## 11      11  217 238    text that is outdated.     TRUE      <NA>      <NA> noname_1
## 12      12  240 264 It will be deleted in the     TRUE      <NA>      <NA> noname_1
## 13      13  266 277              near future.     TRUE      <NA>      <NA> noname_1
## 14      14  280 303  It is important to spell     TRUE      <NA>      <NA> noname_1
## 15      15  305 327   check this dokument. On     TRUE      <NA>      <NA> noname_1
## 16      16  329 345         the other hand, a     TRUE      <NA>      <NA> noname_1
## 17      17  347 367     misspelled word isn't     TRUE      <NA>      <NA> noname_1
## 18      18  369 389     the end of the world.     TRUE      <NA>      <NA> noname_1
## 19      19  391 412    Nothing in the rest of     TRUE      <NA>      <NA> noname_1
## 20      20  414 436   this paragraph needs to     TRUE      <NA>      <NA> noname_1
## 21      21  438 459    be changed. Things can     TRUE      <NA>      <NA> noname_1
## 22      22  461 478        be added after it.     TRUE      <NA>      <NA> noname_1
```

```
dp$
  text_link()$
  text_align()$
  text_data_inherit(
    link      = 1,
    direction = "forward"
  )

dp$tokenize_text_data_lines(2)
```

```
##    token_i from  to                      token is_token     test1     test2     name
## 1        1    1  20     This is an important     TRUE      <NA>      <NA> noname_2
## 2        2   22  38        notice! It should     TRUE      <NA>      <NA> noname_2
## 3        3   40  62  therefore be located at     TRUE      <NA>      <NA> noname_2
## 4        4   64  84    the beginning of this     TRUE      <NA>      <NA> noname_2
## 5        5   86  94                 document!     TRUE      <NA>      <NA> noname_2
## 6        6   97 112          This part of the     TRUE inherited inherited noname_2
## 7        7  114 136  document has stayed the     TRUE inherited inherited noname_2
## 8        8  138 157     same from version to     TRUE inherited inherited noname_2
## 9        9  159 180    version.  It shouldn't     TRUE inherited inherited noname_2
## 10      10  182 203    be shown if it doesn't     TRUE inherited inherited noname_2
```

```
## 11      11  205 228 change.  Otherwise, that     TRUE inherited inherited noname_2
## 12      12  230 252  would not be helping to     TRUE      <NA>      <NA> noname_2
## 13      13  254 271       compress anything.     TRUE      <NA>      <NA> noname_2
## 14      14  274 297 It is important to spell     TRUE      <NA>      <NA> noname_2
## 15      15  299 321  check this document. On     TRUE      <NA>      <NA> noname_2
## 16      16  323 339        the other hand, a     TRUE      <NA>      <NA> noname_2
## 17      17  341 361    misspelled word isn't     TRUE      <NA>      <NA> noname_2
## 18      18  363 383    the end of the world.     TRUE      <NA>      <NA> noname_2
## 19      19  385 406   Nothing in the rest of     TRUE      <NA>      <NA> noname_2
## 20      20  408 430  this paragraph needs to     TRUE      <NA>      <NA> noname_2
## 21      21  432 453   be changed. Things can     TRUE      <NA>      <NA> noname_2
## 22      22  455 472       be added after it.     TRUE      <NA>      <NA> noname_2
## 23      23  475 497  This paragraph contains     TRUE      <NA>      <NA> noname_2
## 24      24  499 521  important new additions     TRUE      <NA>      <NA> noname_2
## 25      25  523 539        to this document.     TRUE      <NA>      <NA> noname_2
```

2.16 Saving and Loading Projects

Diffrprojects also allow for storing and loading project to and from disk.

```
# save to file
dp$save(file = "dp_save.RData")

# remove object
rm(dp)

# create new object and load saved data into new object
dp <- diffrproject$new()
dp$load("dp_save.RData")
dp$tokenize_text_data_lines(2)
```

```
##    token_i from  to                    token is_token     test1     test2     name
## 1        1    1  20     This is an important     TRUE      <NA>      <NA> noname_2
## 2        2   22  38        notice! It should     TRUE      <NA>      <NA> noname_2
## 3        3   40  62  therefore be located at     TRUE      <NA>      <NA> noname_2
## 4        4   64  84    the beginning of this     TRUE      <NA>      <NA> noname_2
## 5        5   86  94                 document!     TRUE      <NA>      <NA> noname_2
## 6        6   97 112         This part of the     TRUE inherited inherited noname_2
## 7        7  114 136  document has stayed the     TRUE inherited inherited noname_2
## 8        8  138 157     same from version to     TRUE inherited inherited noname_2
## 9        9  159 180   version.  It shouldn't     TRUE inherited inherited noname_2
## 10      10  182 203   be shown if it doesn't     TRUE inherited inherited noname_2
## 11      11  205 228 change.  Otherwise, that     TRUE inherited inherited noname_2
## 12      12  230 252  would not be helping to     TRUE      <NA>      <NA> noname_2
## 13      13  254 271       compress anything.     TRUE      <NA>      <NA> noname_2
## 14      14  274 297 It is important to spell     TRUE      <NA>      <NA> noname_2
## 15      15  299 321  check this document. On     TRUE      <NA>      <NA> noname_2
## 16      16  323 339        the other hand, a     TRUE      <NA>      <NA> noname_2
## 17      17  341 361    misspelled word isn't     TRUE      <NA>      <NA> noname_2
## 18      18  363 383    the end of the world.     TRUE      <NA>      <NA> noname_2
## 19      19  385 406   Nothing in the rest of     TRUE      <NA>      <NA> noname_2
## 20      20  408 430  this paragraph needs to     TRUE      <NA>      <NA> noname_2
## 21      21  432 453   be changed. Things can     TRUE      <NA>      <NA> noname_2
## 22      22  455 472       be added after it.     TRUE      <NA>      <NA> noname_2
```

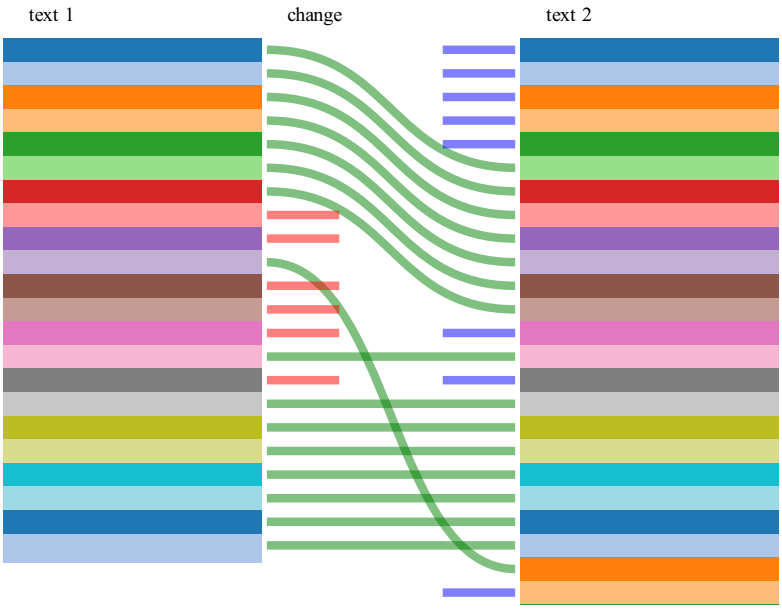
```
## 23      23  475 497  This paragraph contains     TRUE      <NA>      <NA> noname_2
## 24      24  499 521  important new additions     TRUE      <NA>      <NA> noname_2
## 25      25  523 539         to this document.     TRUE      <NA>      <NA> noname_2
```

3 Diffrprojectswidget a Diffrprojects Extension

```
library(diffrprojectswidget)
dp_table(dp, 1, height = 800)
```

#	token_1	#1	type	distance	#2	token_2
[1]	*This part of the*	[1]	==	0	[6]	*This part of the*
[23]			*ins*	20	[1]	*This is an important*
[2]	*document has stayed the*	[2]	==	0	[7]	*document has stayed the*
[24]			*ins*	17	[2]	*notice! It should*
[3]	*same from version to*	[3]	==	0	[8]	*same from version to*
[25]			*ins*	23	[3]	*therefore be located at*
[4]	*version. It shouldn't*	[4]	==	0	[9]	*version. It shouldn't*
[26]			*ins*	21	[4]	*the beginning of this*
[5]	*be shown if it doesn't*	[5]	==	0	[10]	*be shown if it doesn't*
[27]			*ins*	9	[5]	*document!*
[6]	*change. Otherwise, that*	[6]	==	0	[11]	*change. Otherwise, that*
[7]	*would not be helping to*	[7]	==	0	[12]	*would not be helping to*
[8]	*compress the size of the*	[8]	*del*	24		
[9]	*changes.*	[9]	*del*	8		
[10]	*This paragraph contains*	[10]	==	0	[23]	*This paragraph contains*
[11]	*text that is outdated.*	[11]	*del*	22		
[12]	*It will be deleted in the*	[12]	*del*	25		
[13]	*near future.*	[13]	*del*	12		
[28]			*ins*	18	[13]	*compress anything.*
[14]	*It is important to spell*	[14]	==	0	[14]	*It is important to spell*
[15]	*check this dokument. On*	[15]	*del*	23		
[29]			*ins*	23	[15]	*check this document. On*
[16]	*the other hand, a*	[16]	==	0	[16]	*the other hand, a*
[17]	*misspelled word isn't*	[17]	==	0	[17]	*misspelled word isn't*
[18]	*the end of the world.*	[18]	==	0	[18]	*the end of the world.*
[19]	*Nothing in the rest of*	[19]	==	0	[19]	*Nothing in the rest of*
[20]	*this paragraph needs to*	[20]	==	0	[20]	*this paragraph needs to*
[21]	*be changed. Things can*	[21]	==	0	[21]	*be changed. Things can*
[22]	*be added after it.*	[22]	==	0	[22]	*be added after it.*
[30]			*ins*	23	[24]	*important new additions*
[31]			*ins*	17	[25]	*to this document.*

```
library(diffrprojectswidget)
dp_vis(dp, 1, height = 300)
```



4 Technicalities

4.1 Naming Conventions and General Structure of Methods and Data

The methods and data fields of diffrprojects can be categorized into five realms - *cursive*: methods; (parentheses): private; rest: data:

- **text**: everything related to individual texts starts with text
  - text, *text_add*, *text_delete*, *text_align*, *text_code*, *text_code_alignment_token*, *text_code_alignment_token_regex*, *text_code_regex*,
  - *text_data*, *text_data_inherit*, *tokenize_text_data_lines*, *tokenize_text_data_regex*, *tokenize_text_data_words*
  - text_meta_data
- **alignment**: everything that concerns the relation between two texts
  - alignment, *alignment_add*, *alignment_code*, *alignment_delete*, *alignemtn_data_full*, *alignment_data_set*
  - text_link, *link*

- **misc**:
  - meta, options, *load*, *save*, *export_sqlite*, *import_sqlite*, (*execute_load*), (*prepare_save*)
- **inherited from R6_rtext_extended**:
  - options, *message*, *warning*, (*hash*), (hashed), (hashes)
- **inherited from R6**:
  - *clone*, *initialize*

4.2 Data formats

4.2.1 meta

Meta is a list with only a few items providing/storing general information for the whole project - i.e. time stamp the project was created, path to store data, path to export data, an project id.

4.2.2 text

Text is a list of rtext instances. Each rtext instance stores text's actual text as data gathered on the text.

The text_data method will return a data.frame containing all text data, while tokenize_text_data_xxx methods will aggregate text data to specific token levels: words, lines or user defined patterns.

4.2.3 link

Link is a list of links between texts. Link defines for which text combination alignments should be calculated. Each list item hold a from and to field which stores the names of texts to be aligned. The method to create links is text_link, it also allows to delete specific links.

Link data can be transformed to one big data.frame via: as.data.frame function.

4.2.4 alignment

Alignment is a list of data.frames. Each alignment list item stores which part of one text (character span) is connected to which part of another text (character span).

The list of alignments can be transformed to one big data.frame via: as.data.frame function.

4.2.5 alignment_data

The list of alignment_data can be transformed to one big data.frame via: as.data.frame function.

[[[???!!!]]]

4.3 The Diffrprojects Universe

Diffrprojects has two other packages it relies heavily on and one package that adds further features.

4.3.1 Rtext

Rtext is a package providing a data structure and accompanying methods to handle texts / strings / characters as well as data bound to these texts / strings / characters. All string manipulations are based upon the stringb package. All diffrproject texts are actually rtext instances. Unfortunate you cannot yet manipulate rtext objects once they are part of a diffrproject and expect that data on the relation between texts (i.e. alignment and alignment_data) gets updated as well - hence manipulating texts might lead to inconsistencies in alignments and alignment_data.

A strategy to implement such a feature would be to extend rtext in such a way that text manipulation methods would pass change information to a list of call back functions. Furthermore, diffrprojects need two methods that allow for handling shifts in the character sequences of texts. Those update methods can then be passed to rtext instances once they become part of a diffrproject. Then whenever e.g. some characters are deleted, alignments as well as alignment data touching these character spans get deleted as well and character span information for all other alignments get shifted by the appropriate amount.

For those preferring a version using stringi/stringr - go ahead - since rtext and diffrprojects provide tests for all respectively for all vital parts and stringb copied the function naming scheme from stringr anyways, this should be a small matter.

4.3.2 Stringb

Stringb is a package providing convenience functions for string handling and manipulation using R's own regular expression engine. All string manipulations are based upon the stringb package.

In addition stringb provides very flexible text tokenization functions that are very much in line with the needs of diffrprojects.

4.3.3 Diffrprojectswidget

This package enhances diffrprojects by providing HTMLwidgets for visualizing diffrproject data: as interactive table or as interactive graph.

HTMLwidgets (see: http://www.htmlwidgets.org/) are a framework that allows for interactive, web technology based graphics that are furthermore easily integrate able into e.g. R-shiny (http://shiny.rstudio.com/) applications.

4.4 Two words or more about Objects / R6 / Classes / Instances

Diffrprojects is written in object oriented programming style because it seemed adequate to do so. Why? Because in OOP in comparison to functional programming one does more stuff like in-place-modification of data, data and its modifiers (methods) come in one big bundle, it´s easier to work on the current state of the object / to only allow consistent states of the object. Yeap everything here could have done with FP as well - please go ahead.

The downside of using OOP in R is that what happens becomes much more intrasparent and harder to reason about - I am sorry for that.

4.4.1 Classes and Instances

Classes are object blueprints - a schema that describes how an object of this class should look like. Classes might be objects too, but they are not the objects they describe. To get an object instance of an object - a manifestion of the idea of the object described in the class - one has to explicitly translate execute the instructions led out in the class, e.g. via: `diffrproject$new()`, or `rtext$new()`

4.4.2 R6

R6 is a package that provides a framework that makes it very, very easy to build objects in R that are more like things known from traditional all purpose programming languages like Java or C++.

4.4.3 Methods

Methods are exactly like functions, only that they are not floating around loosely in your global environment or elsewhere, but are bound to specific instances of an object. So there is not one text_add function that can be used with any diffrproject, but there is one specific text_add method for each instance of an diffrproject. This sounds quite strange, right? Why the duplication? Well, with that you can e.g. pass this method around, hand it over to a function that calls it or put it into another object maybe that than can decide to use it or not. A silly example:

```
dp1 <- diffrproject$new()
add_text_to_dp1 <- dp1$text_add

add_text_to_dp1("ahhh")
add_text_to_dp1("behhh")
add_text_to_dp1("cehhh")

names(dp1$text)
```

```
## [1] "noname_1" "noname_2" "noname_3"
```