

Package ‘bdpar’

January 9, 2020

Type Package

Title Big Data Preprocessing Architecture

Version 1.0.1

Description Provide a tool to easily build customized data flows to pre-process large volumes of information from different sources. To this end, 'bdpar' allows to (i) easily use and create new functionalities and (ii) develop new data source extractors according to the user needs. Additionally, the package provides by default a predefined data flow to extract and pre-process the most relevant information (tokens, dates, ...) from some textual sources (SMS, Email, tweets, YouTube comments).

Date 2019-12-23

License GPL-3

URL <https://github.com/miferreiro/bdpar>

BugReports <https://github.com/miferreiro/bdpar/issues>

Depends R (>= 3.5.0)

Imports ini, magrittr, pipeR, purrr, R6, rlist, svMisc, tools, utils

Suggests cld2, knitr, readr, rex, rjson, rmarkdown, rtweet, stringi, stringr, testthat (>= 2.1.1), textutils, tuber

VignetteBuilder knitr

RoxygenNote 6.1.1

SystemRequirements Python (>= 2.7)

Encoding UTF-8

NeedsCompilation no

Author Miguel Ferreiro-Díaz [aut, cre],
David Ruano-Ordás [aut, ctr],
Tomás R. Cotos-Yañez [aut, ctr],
University of Vigo [cph]

Maintainer Miguel Ferreiro-Díaz <miguel.ferreiro.diaz@gmail.com>

Repository CRAN

Date/Publication 2020-01-09 14:30:11 UTC

R topics documented:

AbbreviationPipe	2
Bdpar	5
Connections	7
ContractionPipe	9
ExtractorEml	11
ExtractorSms	12
ExtractorTwtid	13
ExtractorYtbid	15
File2Pipe	16
FindEmojiPipe	17
FindEmoticonPipe	18
FindHashtagPipe	20
FindUrlPipe	21
FindUserNamePipe	24
GuessDatePipe	25
GuessLanguagePipe	26
Instance	27
InstanceFactory	30
InterjectionPipe	31
MeasureLengthPipe	33
PipeGeneric	34
pipeline_execute	36
ResourceHandler	38
SerialPipe	39
SlangPipe	40
StopWordPipe	42
StoreFileExtPipe	44
TargetAssigningPipe	45
TeeCSVPipe	47
ToLowerCasePipe	48
TypePipe	49
%>I%	50
Index	52

AbbreviationPipe	<i>Class to find and/or replace the abbreviations on the data field of an Instance</i>
------------------	--

Description

[AbbreviationPipe](#) class is responsible for detecting the existing abbreviations in the **data** field of each [Instance](#). Identified abbreviations are stored inside the **abbreviation** field of [Instance](#) class. Moreover if needed, is able to perform inline abbreviations replacement.

Usage

AbbreviationPipe

Constructor

```
AbbreviationPipe$new(propertyName = "abbreviation",
                    propertyLanguageName = "language",
                    alwaysBeforeDeps = list("GuessLanguagePipe"),
                    notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **propertyLanguageName:** (*character*) name of the language property.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

[AbbreviationPipe](#) class requires the resource files (in json format) containing the correspondence between abbreviations and meaning. To this end, the language of the text indicated in the *propertyLanguageName* should be contained in the resource file name (ie. abbrev.xxx.json where xxx is the value defined in the *propertyLanguageName*). The location of the resources should be defined in the *resourcesPath* section of the configuration file.

[resourcesPath]

```
resourcesAbbreviationsPath = «resources_abbreviations_path»
```

Note

[AbbreviationPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain/replace the abbreviations. The abbreviations found in the Pipe are added to the list of properties of the [Instance](#). If the *replaceAbbreviations* parameter is TRUE, the [Instance](#) data will be modified by replacing the abbreviations found.
 - *Usage:* pipe(instance, replaceAbbreviations = TRUE)
 - *Value:* the [Instance](#) with the modifications that have occurred in the pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **replaceAbbreviations:** (*logical*) indicates if the abbreviations are replaced or not.
- **findAbbreviation:** checks if the abbreviation is in the data.

- *Usage*: findAbbreviation(data, abbreviation)
- *Value*: boolean, depending on whether the abbreviation is in the data.
- *Arguments*:
 - * **data**: (*character*) text where abbreviation will be searched.
 - * **abbreviation**: (*character*) indicates the abbreviation to find.
- **replaceAbbreviation**: replaces the abbreviation in the data for the extendedAbbreviation.
 - *Usage*: replaceAbbreviation(abbreviation, extendedAbbreviation, data)
 - *Value*: the data with the abbreviations replaced.
 - *Arguments*:
 - * **abbreviation**: (*character*) indicates the abbreviation to replace.
 - * **extendedAbbreviation**: (*character*) indicates the string to replace for the abbreviations found.
 - * **data**: (*character*) text where abbreviation will be replaced.
- **getPropertyLanguageName**: gets of name of property language.
 - *Usage*: getPropertyLanguageName()
 - *Value*: value of name of property language.
- **getResourcesAbbreviationsPath**: gets of path of abbreviations resources.
 - *Usage*: getResourcesAbbreviationsPath()
 - *Value*: value of path of abbreviations resources.
- **setResourcesAbbreviationsPath**: sets the path of abbreviations resources.
 - *Usage*: setResourcesAbbreviationsPath(path)
 - *Arguments*:
 - * **path**: (*character*) the new value of the path of abbreviations resources.

Private fields

- **propertyLanguageName**: (*character*) the name of property about language.
- **resourcesAbbreviationsPath**: (*character*) the path where are the resources.

See Also

[ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

Bdpar

Class to manage the preprocess of the files throughout the flow of pipes

Description

Bdpar class provides the static variables required to perform the whole data flow process. To this end Bdpar is in charge of (i) loading the configuration parameters (from configurationsTemplate.ini or the user configuration file) and (ii) executing the flow of pipes (inherited from [TypePipe](#) class) passed as argument.

Usage

Bdpar

Constructor

```
Bdpar$new(configurationFilePath = NULL, editConfigurationFile = FALSE)
```

- *Arguments:*

- **configurationFilePath:** (*character*) path where the configuration file is located. The file must have the .ini extension. In the case that the argument is null, the default configuration file (configurationsTemplate.ini) will be used.
- **editConfigurationFile:** (*boolean*) indicates if open an editor to change the configuration file or not.

Details

The configuration file can be indicated by the user or use the default configuration file (configurationsTemplate.ini). In addition, once we call the Bdpar constructor, it will be possible to choose if the user wants to edit the file of indicated configurations or not.

The *configurationFilePath* file should have the following structure (Depends on the Pipes used). Also the configurationsTemplate.ini has this structure:

[twitter]

```
ConsumerKey = «consumer_key»
```

```
ConsumerSecret = «consumer_secret»
```

```
AccessToken = «access_token»
```

```
AccessTokenSecret = «access_token_secret»
```

[youtube]

```
app_id = «app_id»
```

```
app_password = «app_password»
```

[eml]

```
PartSelectedOnMPAlternative = «part_selected» (text/html or text/plain)
```

[resourcesPath]

```
resourcesAbbreviationsPath = «resources_abbreviations_path»
resourcesContractionsPath = «resources_contractions_path»
resourcesInterjectionsPath = «resources_interjections_path»
resourcesSlangsPath = «resources_slangs_path»
resourcesStopWordsPath = «resources_stopWords_path»
```

[CSVPath]

```
outPutTeeCSVPipePath = «out_put_teeCSVPipe_path»
```

[cache]

```
cachePathTwtid = «cache_path_twtid»
cachePathYtbid = «cache_path_ytbid»
```

Note

In the case of choosing to edit the configuration file, the default editor will be opened.

Static variables

- **connections:** (*Connections*) object that handles the connections with YouTube and Twitter.
- **configurationFilePath:** (*character*) path where the configuration file is located. The file must have the .ini extension. In the case that the argument is null, the default configuration file (configurationsTemplate.ini) will be used. ,
- **resourceHandler:** (*ResourceHandler*) object that handles the json resources files.

Methods

- **process_files:** preprocess files through the indicated flow of pipes.
 - *Usage:*

```
process_files(filesPath,
              pipe = SerialPipe$new(),
              instanceFactory = InstanceFactory$new())
```
 - *Value:* list of Instances that have been preprocessed.
 - *Arguments:*
 - * **filesPath:** (*character*) path where the files to be processed are located.
 - * **pipe:** (*TypePipe*) subclass of [TypePipe](#), which implements the pipe method.
 - * **instanceFactory:** (*InstanceFactory*) class which implements the method `createInstance` to choose which type of [Instance](#) is created.

See Also

[Connections](#), [Instance](#), [InstanceFactory](#), [pipeline_execute](#), [TypePipe](#), [SerialPipe](#)

Examples

```
## Not run:
#Path where the configuration file are located
configurationFilePath <- system.file(file.path("examples",
                                             "configurationsExample.ini"),
                                   package = "bdpar")

#Folder with the files to preprocess
filesPath <- system.file(file.path("examples",
                                   "testFiles"),
                        package = "bdpar")

#Object which indicates the pipes' flow
pipe <- SerialPipe$new()

#Object which decides how creates the instances
instanceFactory <- InstanceFactory$new()

objectBdpar <- Bdpar$new(configurationFilePath)

#Starting file preprocessing...
objectBdpar$process_files(filesPath, pipe, instanceFactory)

## End(Not run)
```

Connections

Class to manage the connections with Twitter and YouTube

Description

The tasks of the functions that the [Connections](#) class has are to establish the connections and control the number of requests that have been made with the APIs of Twitter and YouTube.

Usage

Connections

Constructor

Connections\$new(keysPath)

- *Arguments:*

- **keysPath:** (*character*) path of the .ini file that contains the keys.

Details

The way to indicate the keys of YouTube and Twitter has to be through the configuration file that contains the following structure:

[twitter]

ConsumerKey = «*consumer_key*»

ConsumerSecret = «*consumer_secret*»

AccessToken = «*access_token*»

AccessTokenSecret = «*access_token_secret*»

[youtube]

app_id = «*app_id*»

app_password = «*app_password*»

Note

Fields of unused connections will be automatically ignored by the platform.

Methods

- **getTwitterToken:** gets the Twitter token ID.
 - *Usage:* getTwitterToken()
 - *Value:* value of twitterToken.
- **startConnectionWithTwitter:** is responsible of establishing the connection to Twitter.
 - *Usage:* startConnectionWithTwitter()
- **checkRequestToTwitter** function in charge of handling the connection with Twitter.
 - *Usage:* checkRequestToTwitter()
- **startConnectionWithYoutube** function able to establish the connection with YouTube.
 - *Usage:* startConnectionWithYoutube()
- **addNumRequestToYoutube** function that increases in one the number of request to YouTube.
 - *Usage:* addNumRequestToYoutube()
- **checkRequestToYoutube** handles the connection with YouTube.
 - *Usage:* checkRequestToYoutube()
- **getNumRequestMaxToYoutube** gets the number of maximum requests allowed by YouTube API.
 - *Usage:* getNumRequestMaxToYoutube()
 - *Value:* value of number maximum of request to YouTube.

Private fields

- **keys:** (*list*) the keys of Twitter and YouTube.
- **numRequestToYoutube:** (*numeric*) indicates the number of requests made to YouTube.
- **numRequestMaxToYoutube:** (*numeric*) indicates the maximum number of requests with YouTube.
- **connectionWithYoutube:** (*logical*) indicates if the connection has been established with YouTube.
- **connectionWithTwitter:** (*logical*) indicates if the connection has been established with Twitter.
- **twitterToken:** (*Token*) token to establish the connection to Twitter.

See Also

[ExtractorTwtid](#), [ExtractorYtbid](#)

ContractionPipe	<i>Class to find and/or replace the contractions on the data field of a Instance</i>
-----------------	--

Description

[ContractionPipe](#) class is responsible for detecting the existing contractions in the **data** field of each [Instance](#). Identified contractions are stored inside the **contraction** field of [Instance](#) class. Moreover if needed, is able to perform inline contractions replacement.

Usage

```
ContractionPipe
```

Constructor

```
ContractionPipe$new(propertyName = "contractions",
                    propertyLanguageName = "language",
                    alwaysBeforeDeps = list("GuessLanguagePipe"),
                    notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **propertyLanguageName:** (*character*) name of the language property.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

`ContractionPipe` class requires the resource files (in json format) containing the correspondence between contractions and meaning. To this end, the language of the text indicated in the `propertyLanguageName` should be contained in the resource file name (ie. `contr.xxx.json` where `xxx` is the value defined in the `propertyLanguageName`). The location of the resources should be defined in the `resourcesPath` section of the configuration file.

[resourcesPath]

`resourcesContractionsPath = «resources_contractions_path»`

Note

`ContractionPipe` will automatically invalidate the `Instance` whenever the obtained data is empty.

Inherit

This class inherits from `PipeGeneric` and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the `Instance` to obtain/replace the contractions. The contractions found in the Pipe are added to the list of properties of the `Instance`. If the `replaceContractions` parameter is `TRUE`, the `Instance` data will be modified by replacing the contractions found.
 - *Usage*: `pipe(instance, replaceContractions = TRUE)`
 - *Value*: the `Instance` with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) `Instance` to preprocess.
 - * **replaceContractions**: (*logical*) indicates if the contractions are replaced or not.
- **findContraction**: checks if the contraction is in the data.
 - *Usage*: `findContraction(data, contraction)`
 - *Value*: boolean, depending on whether the contraction is on the data.
 - *Arguments*:
 - * **data**: (*character*) text where contraction will be searched.
 - * **contraction**: (*character*) indicates the contraction to find.
- **replaceContraction**: replaces the contraction in the data for the *extendedContraction*.
 - *Usage*: `replaceContraction(contraction, extendedContraction, data)`
 - *Value*: the data with the contractions replaced.
 - *Arguments*:
 - * **contraction**: (*character*) indicates the contraction to remove.
 - * **extendedContraction**: (*character*) indicates the string to replace for the contraction found.
 - * **data**: (*character*) text where contraction will be replaced.
- **getPropertyLanguageName**: gets the name of property language.
 - *Usage*: `getPropertyLanguageName()`
 - *Value*: value of name of property language.

- **getResourcesContractionsPath:** gets of path of contractions resources.
 - *Usage:* getResourcesContractionsPath()
 - *Value:* value of path of contractions resources.
- **setResourcesContractionsPath:** sets the path of contractions resources.
 - *Usage:* setResourcesContractionsPath(path)
 - *Arguments:*
 - * **path:** (*character*) the new value of the path of contractions resources.

Private fields

- **propertyLanguageName:** (*character*) the name of property about language.
- **resourcesContractionsPath:** (*character*) the path where are the resources.

See Also

[AbbreviationPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

ExtractorEml

Class to handle email files with eml extension

Description

This class inherits from the [Instance](#) class and implements the functions of extracting the text and the date from an eml type file.

Usage

ExtractorEml

Constructor

ExtractorEml\$new(path)

- *Arguments:*
 - **path:** (*character*) path of the eml type file.

Details

The way to indicate which part to choose in the email, when is a multipart email, is through the configuration file which contains the following structure (being YourPartSelected, text/plain or text/html):

[eml]

PartSelectedOnMPAlternative = YourPartSelected (text/html or text/plain)

Note

To be able to use this class it is necessary to have Python installed.

Inherit

This class inherits from [Instance](#) and implements the `obtainSource` and `obtainDate` abstracts functions.

Methods

- **obtainDate:** obtains the date of the eml file. Calls the function `read_emails` and obtains the date of the file indicated in the path and then transforms it into the generic date format, that is `"%a %b %d %H:%M:%S %Z %Y"` (Example: "Thu May 02 06:52:36 UTC 2013").
 - *Usage:* `obtainDate()`
- **obtainSource:** obtains the source of the eml file. Calls the function `read_emails` and obtains the source of the file indicated in the path. In addition, it initializes the data with the initial source.
 - *Usage:* `obtainSource()`
- **getPartSelectedOnMPAlternative:** gets of `PartSelectedOnMPAlternative` variable.
 - *Usage:* `getPartSelectedOnMPAlternative()`
 - *Value:* value of `PartSelectedOnMPAlternative` variable.
- **setPartSelectedOnMPAlternative:** sets of `PartSelectedOnMPAlternative` variable.
 - *Usage:* `setPartSelectedOnMPAlternative(PartSelectedOnMPAlternative)`
 - *Arguments:*
 - * **PartSelectedOnMPAlternative** (*character*) the new value of `PartSelectedOnMPAlternative` variable.

Private fields

- **PartSelectedOnMPAlternative:** (*character*) configuration to read the eml files. Indicates whether the text/plain part or the text/html part is read in multipart emails.

See Also

[ExtractorSms](#), [ExtractorTwtid](#), [ExtractorYtbid](#), [Instance](#)

ExtractorSms

Class to handle SMS files with tsms extension

Description

This class that inherits from the [Instance](#) class and implements the functions of extracting the text and the date of an tsms type file.

Usage

ExtractorSms

Constructor

ExtractorSms\$new(path)

- *Arguments:*
 - **path:** (*character*) path of the tsms type file.

Details

Due to the fact that the creation date of the message can not be extracted from the text of an SMS, the date will be initialized to empty.

Inherit

This class inherits from [Instance](#) and implements the `obtainSource` and `obtainDate` abstracts functions.

Methods

- **obtainDate:** function that obtains the date of the SMS file.
 - *Usage:* `obtainDate()`
- **obtainSource:** obtains the source of the SMS file. Reads the file indicated in the path. In addition, it initializes the data with the initial source.
 - *Usage:* `obtainSource()`

See Also

[ExtractorEml](#), [ExtractorTwtid](#), [ExtractorYtbid](#), [Instance](#)

ExtractorTwtid

Class to handle tweets files with twtid extension

Description

This class inherits from the [Instance](#) class and implements the functions of extracting the text and the date of an twtid type file.

Usage

ExtractorTwtid

Constructor

ExtractorTwtid\$new(path)

- *Arguments:*
 - **path:** (*character*) path of the twtid type file.

Details

Twitter connection is handled through the [Connections](#) class which loads the Twitter API credentials from the configuration file. Additionally, to increase the processing speed, each twitter query is stored in a cache to avoid the execution of duplicated queries. To enable this option, cache location should be in the *cachePathTwtid* indicated in the *cache* section from the configuration file. This variable has to be the path to store the tweets and it is necessary that it has two folder named: "_spam_" and "_ham_"

[cache]

cachePathTwtid = «*cache_path_twtid*»

Inherit

This class inherits from [Instance](#) and implements the `obtainSource` and `obtainDate` abstracts functions.

Methods

- **obtainId:** obtains the ID of an specific tweet. Reads the ID of the file indicated in the variable path.
 - *Usage:* `obtainId()`
- **getId:** gets the ID of an specific tweet.
 - *Usage:* `getId()`
 - *Value:* value of tweet ID.
- **obtainDate:** obtains the date from a specific tweet ID. If the tweet has been previously cached the tweet date is loaded from cache path. Otherwise, the request is performed using Twitter API and the date is automatically formatted to "
 - *Usage:* `obtainDate()`
- **obtainSource:** obtains the source from a specific tweet ID. If the tweet has previously been cached the source is loaded from cache path. Otherwise, the request is performed using on Twitter API.
 - *Usage:* `obtainSource()`

Private fields

- **id:** (*character*) ID of tweet.

See Also

[ExtractorEml](#), [ExtractorSms](#), [ExtractorYtbid](#), [Instance](#),

ExtractorYtbid

Class to handle comments of YouTube files with ytbid extension

Description

This class inherits from the [Instance](#) class and implements the functions of extracting the text and the date of an ytbid type file.

Usage

ExtractorYtbid

Constructor

ExtractorYtbid\$new(path)

- *Arguments:*
 - **path:** (*character*) path of the ytbid type file.

Details

YouTube connection is handled through the [Connections](#) class which loads the YouTube API credentials from the configuration file. Additionally, to increase the processing speed, each youtube query is stored in a cache to avoid the execution of duplicated queries. To enable this option, cache location should be in the *cachePathYtbid* indicated in the *cache* section from the configuration file. This variable has to be the path to store the comments and it is necessary that it has two folder named: "_spam_" and "_ham_"

[cache]

cachePathYtbid = «*cache_path_ytbid*»

Inherit

This class inherits from [Instance](#) and implements the `obtainSource` and `obtainDate` abstracts functions.

Methods

- **obtainId:** obtains the id of the ytbid. Read the id of the file indicated in the variable path.
 - *Usage:* `obtainId()`
- **getId:** gets of comment ID.
 - *Usage:* `getId()`
 - *Value:* value of comment ID.
- **obtainDate:** obtains the date from a specific comment ID. If the comment has been previously cached the comment date is loaded from cache path. Otherwise, the request is performed using YouTube API and the date is then formatted to the established standard.

- *Usage:* obtainDate()
- **obtainSource:** obtains the source from a specific comment ID. If the comment has previously been cached the source is loaded from cache path. Otherwise, the request is performed using on YouTube API.
 - *Usage:* obtainSource()

Private fields

- **id:** (*character*) ID of comment.

See Also

[ExtractorEml](#), [ExtractorSms](#), [ExtractorTwtid](#), [Instance](#)

File2Pipe

Class to obtain the source field of an Instance

Description

Obtains the **source** using the method which implements the subclass of [Instance](#).

Usage

File2Pipe

Constructor

```
File2Pipe$new(propertyName = "source",
               alwaysBeforeDeps = list("TargetAssigningPipe"),
               notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Note

[File2Pipe](#) will automatically invalidate the [Instance](#) whenever the obtained source is empty or not in UTF-8 format.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the [Instance](#) to obtain the source.
 - *Usage*: pipe(instance)
 - *Value*: the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

FindEmojiPipe	<i>Class to find and/or replace the emoji on the data field of an Instance</i>
---------------	--

Description

This class is responsible of detecting the existing emojis in the **data** field of each [Instance](#). Identified emojis are stored inside the **emoji** field of [Instance](#) class. Moreover if required, is able to perform inline emoji replacement.

Usage

```
FindEmojiPipe
```

Constructor

```
FindEmojiPipe$new(propertyName = "emoji",
                  alwaysBeforeDeps = list(),
                  notAfterDeps = list())
```

- *Arguments*:
 - **propertyName**: (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps**: (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps**: (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

[FindEmojiPipe](#) use the emoji list provided by rtweet package.

Note

[FindEmojiPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain/replace the emojis.
 - *Usage:* pipe(instance, replaceEmoji = TRUE)
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **replaceEmoji:** (*logical*) indicates if the emojis are replaced.
- **findEmoji:** checks for the existence of an specific emoji.
 - *Usage:* findEmoji(data, emoji)
 - *Value:* boolean, depending on whether the emoji is on the data.
 - *Arguments:*
 - * **data:** (*character*) text to search the emoji.
 - * **emoji:** (*character*) indicates the emoji to find.
- **replaceEmoji:** replaces the emoji in the data for the extendedEmoji.
 - *Usage:* replaceEmoji(emoji, extendedEmoji, data)
 - *Value:* the data with emoji replaced.
 - *Arguments:*
 - * **emoji:** (*character*) indicates the emoji to remove.
 - * **extendedEmoji:** (*character*) determines the text source to replace the emoji found.
 - * **data:** (*character*) text where emojis will be replaced.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

FindEmoticonPipe

Class to find and/or remove the emoticons on the data field of an Instance

Description

This class is responsible of detecting the existing emoticons in the **data** field of each [Instance](#). Identified emoticons are stored inside the **emoticon** field of [Instance](#) class. Moreover if required, is able to perform inline emoticon removal.

Usage

FindEmoticonPipe

Constructor

```
FindEmoticonPipe$new(propertyName = "emoticon",
                      alwaysBeforeDeps = list(),
                      notAfterDeps = list("FindHashtagPipe"))
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The regular expression indicated in the `emoticonPattern` variable is used to identify emoticons.

Note

`FindEmoticonPipe` will automatically invalidate the `Instance` whenever the obtained data is empty.

Inherit

This class inherits from `PipeGeneric` and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the `Instance` to obtain/remove the emoticons.
 - *Usage:* `pipe(instance, removeEmoticon = TRUE)`
 - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) `Instance` to preprocess.
 - * **removeEmoticon:** (*logical*) indicates if the emoticons are replaced.
- **findEmoticon:** finds the emoticons in the data.
 - *Usage:* `findEmoticon(data)`
 - *Value:* list with emoticons found.
 - *Arguments:*
 - * **data:** (*character*) text to search the emoticons.
- **removeEmoticon:** removes the emoticons in the data.
 - *Usage:* `removeEmoticon(data)`
 - *Value:* the data with emoticons removed.
 - *Arguments:*
 - * **data:** (*character*) text in which emoticons will be removed.

Public fields

- **emoticonPattern:** (*character*) regular expression to detect emoticons.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

FindHashtagPipe	<i>Class to find and/or remove the hashtags on the data field of an Instance</i>
-----------------	--

Description

This class is responsible of detecting the existing hashtags in the **data** field of each [Instance](#). Identified hashtags are stored inside the **hashtag** field of [Instance](#) class. Moreover if required, is able to perform inline hashtag removal.

Usage

```
FindHashtagPipe
```

Constructor

```
FindHashtagPipe$new(propertyName = "hashtag",
                    alwaysBeforeDeps = list(),
                    notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The regular expression indicated in the `hashtagPattern` variable is used to identify hashtags.

Note

[FindHashtagPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the [Instance](#) to obtain/remove the hashtags.
 - *Usage*: `pipe(instance, removeHashtag = TRUE)`
 - *Value* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.
 - * **removeHashtag**: (*logical*) indicates if the hashstags are removed
- **findHashtag**: finds the hashtags in the data.
 - *Usage*: `findHashtag(data)`
 - *Value*: list with hashtags found.
 - *Arguments*:
 - * **data**: (*character*) text to search the hashtags.
- **removeHashtag**: removes the hashtags in the data.
 - *Usage*: `removeHashtag(data)`
 - *Value*: the data with hashtags removed.
 - *Arguments*:
 - * **data**: (*character*) text to remove the hashtags.

Public fields

- **hashtagPattern**: (*character*) regular expression to detect hashtags.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

FindUrlPipe

Class to find and/or remove the URLs on the data field of an Instance

Description

This class is responsible of detecting the existing URLs in the **data** field of each [Instance](#). Identified URLs are stored inside the **URLs** field of [Instance](#) class. Moreover if required, is able to perform inline URLs removal.

Usage

FindUrlPipe

Constructor

```
FindUrlPipe$new(propertyName = "URLs",
                alwaysBeforeDeps = list(),
                notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The regular expressions indicated in the URLPatterns variable are used to identify URLs.

Note

[FindUrlPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain/remove the users.
 - *Usage:*

```
pipe(instance,
      removeUrl = TRUE,
      URLPatterns = list(self$URLPattern, self$EmailPattern),
      namesURLPatterns = list("UrlPattern", "EmailPattern"))
```
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **removeUrl:** (*logical*) indicates if the URLs are removed.
 - * **URLPatterns:** (*list*) the regex to find URLs.
 - * **namesURLPatterns:** (*list*) the names of regex.
- **findUrl:** finds the URLs in the data.
 - *Usage:* findHashtag(pattern, data)
 - *Value:* list with URLs found.
 - *Arguments:*
 - * **pattern:** (*character*) regex to find URLs.
 - * **data:** (*character*) text to search the URLs.

- **removeUrl:** removes the URLs in the data.
 - *Usage:* removeUrl(pattern, data)
 - *Value:* the data with URLs removed.
 - *Arguments:*
 - * **pattern:** (character) regex to find URLs.
 - * **data:** (*character*) text to remove the URLs.
- **putNamesURLPattern:** sets the names to URL patterns result.
 - *Usage:* putNamesURLPattern(resultOfURLPatterns)
 - *Value:* Value of resultOfURLPatterns variable with the names of URL pattern.
 - *Arguments:*
 - * **resultOfURLPatterns:** (*list*) list with URLs found.
- **getURLPatterns:** gets of URL patterns.
 - *Usage:* getURLPatterns()
 - *Value:* value of URL patterns.
- **getNamesURLPatterns:** gets of name of URLs.
 - *Usage:* getNamesURLPatterns()
 - *Value:* value of name of URLs.
- **setNamesURLPatterns:** sets the name of URLs.
 - *Usage:* setNamesURLPatterns(namesURLPatterns)
 - *Arguments:*
 - * **namesURLPatterns:** (*character*) the new value of the name of URLs.

Public fields

- **URLPattern:** (*character*) regular expression to detect URLs.
- **EmailPattern:** (*character*) regular expression to detect emails.

Private fields

- **URLPatterns:** (*list*) regular expressions used to detect URLs.
- **namesURLPatterns:** (*list*) names of regular expressions that are used to identify URLs.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

FindUserNamePipe	<i>Class to find and/or remove the users on the data field of an Instance</i>
------------------	---

Description

This class is responsible of detecting the existing use names in the **data** field of each [Instance](#). Identified user names are stored inside the **userName** field of [Instance](#) class. Moreover if required, is able to perform inline user name removal.

Usage

```
FindUserNamePipe
```

Constructor

```
FindUserNamePipe$new(propertyName = "userName",
                      alwaysBeforeDeps = list(),
                      notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The regular expressions indicated in the `userPattern` variable are used to identify user names.

Note

[FindUserNamePipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain/remove the name users.
 - *Usage:* `pipe(instance, removeUser = TRUE)`
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **removeUser:** (*logical*) indicates if the users are removed.

- **findUserName:** finds the name users in the data.
 - *Usage:* findHashtag(data)
 - *Value:* list with users names found.
 - *Arguments:*
 - * **data:** (*character*) text to search the user names.
- **removeUserName:** removes the users in the data.
 - *Usage:* removeUserName(data)
 - *Value:* the data with name users removed.
 - *Arguments:*
 - * **data:** (*character*) text to remove the user names.

Public fields

- **userPattern:** (*character*) regular expression to detect users.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

GuessDatePipe	<i>Class to obtain the date field of an Instance</i>
---------------	--

Description

Obtains the **date** using the method which implements the subclass of [Instance](#)

Usage

```
GuessDatePipe
```

Constructor

```
GuessDatePipe$new(propertyName = "date",
                  alwaysBeforeDeps = list("TargetAssigningPipe"),
                  notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Inherit

this class inherit from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the [Instance](#) to obtain the date.
 - *Usage*: pipe(instance)
 - *Value*: the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

GuessLanguagePipe	<i>Class to guess the language of an Instance</i>
-------------------	---

Description

This class allows guess the language by using language detector of library cld2. Creates the **language** property which indicates the idiom text. Optionally, it is possible to choose the language provided by Twitter.

Usage

```
GuessLanguagePipe
```

Constructor

```
GuessLanguagePipe$new(propertyName = "language",
                        alwaysBeforeDeps = list("StoreFileExtPipe",
                                                "TargetAssigningPipe"),
                        notAfterDeps = list())
```

- *Arguments*:
 - **propertyName**: (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps**: (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps**: (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

To obtain the language of the tweets, it will be verified that there is a json file with the information stored in memory.

Note

The Pipe will invalidate the [Instance](#) if the language of the data can not be detect.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain the language of the data.
 - *Usage:* pipe(instance, languageTwitter = TRUE)
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **languageTwitter:** (*logical*) indicates whether for the Instances of type twtid the language that returns the api is obtained or the detector is applied.
- **getLanguage:** guesses the language of data.
 - *Usage:* getLanguage(data)
 - *Value:* the language guesser. Format: see ISO 639-3:2007.
 - *Arguments:*
 - * **data:** (*character*) text to guess the language.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

 Instance

Abstract super class that handles the management of the Instances

Description

Provides the required methods to successfully handle each [Instance](#) class.

Usage

Instance

Constructor

Instance\$new(path)

- *Arguments:*
 - **path:** (*character*) path of the file.

Methods

- **obtainDate:** abstract function responsible for obtaining the date of the [Instance](#).
- **obtainSource:** abstract function in charge of determining the source of the [Instance](#).
- **getDate:** gets of date.
 - *Usage:* getDate()
 - *Value:* Value of date.
- **setDate:** sets of date.
 - *Usage:* setDate(date)
 - *Arguments:*
 - * **date:** (*character*) the new value of date.
- **getSource:** gets of source.
 - *Usage:* getSource()
 - *Value:* value of source.
- **setSource:** modifies the source value.
 - *Usage:* setSource(source)
 - *Arguments:*
 - * **source:** (*character*) the new value of source.
- **getPath:** gets of path.
 - *Usage:* getPath()
 - *Value:* value of path.
- **getProperties:** gets the list of properties.
 - *Usage:* getProperties()
 - *Value:* Value of properties.
- **setProperties:** modifies the list of properties.
 - *Usage:* setProperties(properties)
 - *Arguments:*
 - * **properties:** (*list*) containing the new properties.
- **addProperties:** adds a property to the list of properties.
 - *Usage:* addProperties(propertyValue, propertyName)
 - *Arguments:*
 - * **propertyValue:** (*Object*) the value of the new property.
 - * **propertyName:** (*character*) the name of the new property.
- **getSpecificProperty:** obtains a specific property.

- *Usage*: `getSpecificProperty(propertyName)`
- *Value*: the value of the specific property.
- *Arguments*:
 - * **propertyName**: (*character*) the name of the property to obtain.
- **isSpecificProperty**: checks for the existence of an specific property.
 - *Usage*: `isSpecificProperty(propertyName)`
 - *Value*: A boolean results according to the existence of the specific property in the list of properties.
 - *Arguments*:
 - * **propertyName**: (*character*) the name of the property to check.
- **setSpecificProperty**: modifies the value of the one property.
 - *Usage*: `setSpecificProperty(propertyName,propertyValue)`
 - *Arguments*:
 - * **propertyName**: (*Object*) the new value of the property.
 - * **propertyValue**: (*character*) the name of the property.
- **getNamesOfProperties**: gets of the names of all properties.
 - *Usage*: `getNamesOfProperties()`
 - *Value*: the names of properties.
- **isInstanceValid**: checks if the [Instance](#) is valid.
 - *Usage*: `isInstanceValid()`
 - *Value*: value of `isValid`.
- **invalidate**: forces the invalidation of an specific [Instance](#).
 - *Usage*: `invalidate()`
- **getFlowPipes**: gets the list of the flow of Pipes.
 - *Usage*: `getNamesOfProperties()`
 - *Value*: names of the Pipes used.
- **addFlowPipes**: adds a new Pipe to the flow of Pipes.
 - *Usage*: `addFlowPipes(namePipe)`
 - *Arguments*:
 - * **namePipe**: (*character*) name of the new Pipe to be added in the Pipe flow.
- **getBanPipes**: gets an array with containing all the Pipes.
 - *Usage*: `getBanPipes()`
 - *Value*: value of Pipe ban array.
- **addBanPipes**: added the name of the Pipe to the array that keeps the track of Pipes having running after restrictions.
 - *Usage*: `addBanPipes(namePipe)`
 - *Arguments*:
 - * **namePipe**: (*character*) Pipe name to be introduced into the ban array.
- **checkCompatibility**: Check compability between Pipes.
 - *Usage*: `checkCompatibility(namePipe,alwaysBefore)`

- *Value:* boolean, depends if the compability between Pipes is correctly or not.
- *Arguments:*
 - * **namePipe:** (*character*) name of the Pipe to check the compatibility.
 - * **alwaysBefore:** (*list*) pipes that the Instance had to go through.

Private fields

- **date:** (*character*) the date on which the source was generated or sent.
- **source:** (*character*) the text of the file without modifications.
- **path:** (*character*) identifier of the Instance, in this case it will be the path of the file from which the properties are extracted.
- **data:** (*character*) the text of the file with modifications.
- **properties:** (*list*) contains a list of properties extracted from the text that is being processed.
- **isValid:** (*logical*) indicates if the [Instance](#) is valid or not.
- **flowPipes:** (*list*) the list contains the Pipes that the Instance has passed through.
- **banPipes:** (*array*) the list contains the Pipes that can not be executed from that moment.

InstanceFactory

Class to handle the creation of Instance types

Description

[InstanceFactory](#) class builds the appropriate [Instance](#) object according to the file extension.

Usage

```
InstanceFactory
```

Constructor

```
InstanceFactory$new()
```

Methods

- **createInstance:** builds the [Instance](#) object according to the file extension.
 - *Usage:* createInstance(path)
 - *Value:* the [Instance](#) corresponding object according to the file extension.
 - *Arguments:*
 - * **path:** (*character*) path of the file to create an [Instance](#).

See Also

[ExtractorEml](#), [ExtractorSms](#), [ExtractorTwtid](#), [ExtractorYtbid](#), [Instance](#)

InterjectionPipe	<i>Class to find and/or remove the interjections on the data field of an Instance</i>
------------------	---

Description

[InterjectionPipe](#) class is responsible for detecting the existing interjections in the **data** field of each [Instance](#). Identified interjections are stored inside the **interjection** field of [Instance](#) class. Moreover if needed, is able to perform inline interjections removal.

Usage

```
InterjectionPipe
```

Constructor

```
InterjectionPipe$new(propertyName = "interjection",  
                      propertyLanguageName = "language",  
                      alwaysBeforeDeps = list("GuessLanguagePipe"),  
                      notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **propertyLanguageName:** (*character*) name of the language property.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

[InterjectionPipe](#) class requires the resource files (in json format) containing the list of interjections. To this end, the language of the text indicated in the *propertyLanguageName* should be contained in the resource file name (ie. interj.xxx.json where xxx is the value defined in the *propertyLanguageName*). The location of the resources should be defined in the *resourcesPath* section of the configuration file.

[resourcesPath]

```
resourcesInterjectionsPath = «resources_interjections_path»
```

Note

[InterjectionPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe** Preprocesses the [Instance](#) to obtain/remove the interjections. The interjections found in the Pipe are added to the list of properties of the [Instance](#). If the `removeInterjections` parameter is `TRUE`, the interjections in the data will be removed.
 - *Usage:*
`pipe(instance, removeInterjections = TRUE)`
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **removeInterjections:** (*logical*) indicates if the interjections are removed or not.
- **findInterjection:** checks if the interjection is in the data.
 - *Usage:* `findInterjection(data, interjection)`
 - *Value:* boolean, depending on whether the interjection is on the data.
 - *Arguments:*
 - * **data:** (*character*) text where interjection will be replaced.
 - * **interjection:** (*character*) indicate the interjection to find.
- **removeInterjection:** removes the interjection in the data.
 - *Usage:*
`removeInterjection(interjection, data)`
 - *Value:* the data with interjection removed.
 - *Arguments:*
 - * **interjection:** (*character*) indicates the interjection to remove.
 - * **data:** (*character*) text where interjection will be removed.
- **getPropertyLanguageName:** gets of name of property language.
 - *Usage:* `getPropertyLanguageName()`
 - *Value:* value of name of property language.
- **getResourcesInterjectionsPath:** gets of path of interjections resources.
 - *Usage:* `getResourcesInterjectionsPath()`
 - *Value:* value of path of interjections resources.
- **setResourcesInterjectionsPath:** sets the path of interjections resources.
 - *Usage:* `setResourcesInterjectionsPath(path)`
 - *Arguments:*
 - * **path:** (*character*) the new value of the path of interjections resources.

Private fields

- **propertyLanguageName:** (*character*) the name of property about language.
- **resourcesInterjectionsPath:** (*character*) the path where are the resources.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

MeasureLengthPipe	<i>Class to obtain the length of the data field of an Instance</i>
-------------------	--

Description

This class is responsible of obtain the length of the **data** field of each [Instance](#). Creates the **length** property which indicates the length of the text. The property's name is customize through the class constructor.

Usage

```
MeasureLengthPipe
```

Constructor

```
MeasureLengthPipe$new(propertyName = "length",
                        alwaysBeforeDeps = list(),
                        notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain the length of data.
 - *Usage:* `pipe(instance, propertyName = super$getPropertyName(), nchar_conf = TRUE)`
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **propertyName** (*character*) the name of the property that will be obtained in the Pipe.
 - * **nchar_conf** (*logical*) indicates if the Pipe uses nchar or object.size.

- **getLength**: obtains the length of the data.
 - *Usage*: `getLength(data, nchar_conf = TRUE)`
 - *Value*: the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **data**: (*character*) text to preprocess.
 - * **nchar_conf**: (*logical*) indicates if the Pipe uses `nchar` or `object.size`.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

PipeGeneric

Abstract super class that handles the management of the Pipes

Description

Provides the required methods to successfully handle each [PipeGeneric](#) class.

Usage

```
PipeGeneric
```

Constructor

```
PipeGeneric$new(propertyName,
                 alwaysBeforeDeps,
                 notAfterDeps)
```

- *Arguments*:
 - **propertyName**: (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps**: (*list*) the dependencies alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps**: (*list*) the dependencies notAfter (Pipes that cannot be executed after this one).

Methods

- **pipe**: abstract method to preprocess the [Instance](#).
 - *Usage*: `pipe(instance)`
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.
- **getPropertyName**: gets of name of property.

- *Usage:* `getPropertyName()`
- *Value:* value of name of property.
- **getAlwaysBeforeDeps:** gets of the dependences always before.
 - *Usage:* `getAlwaysBeforeDeps()`
 - *Value:* Value of dependences always before.
- **getNotAfterDeps:** gets of the dependences not after.
 - *Usage:* `getNotAfterDeps()`
 - *Value:* value of dependences not after.
- **setPropertyName:** changes the value of property's name.
 - *Usage:* `setPropertyName(propertyName)`
 - *Arguments:*
 - * **propertyName:** (*character*) the new value of the property's name.
- **setAlwaysBeforeDeps:** changes the value of dependencies always before.
 - *Usage:* `setAlwaysBeforeDeps(alwaysBeforeDeps)`
 - *Arguments:*
 - * **alwaysBeforeDeps:** (*list*) the new value of the dependencies always before.
- **setNotAfterDeps:** changes the value of dependencies not after.
 - *Usage:* `setNotAfterDeps(notAfterDeps)`
 - *Arguments:*
 - * **notAfterDeps:** (*list*) the new value of the dependencies not after.

Private fields

- **propertyName:** (*character*) the name of property.
- **alwaysBeforeDeps:** (*list*) dependencies of the type `alwaysBefore`. These dependences indicate what Pipes must be executed before the current one.
- **notAfterDeps:** (*list*) dependencies of the type `notAfter`. These dependences indicate what Pipes must not be executed after the current one.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

pipeline_execute *Initiates the pipelining process.*

Description

pipeline_execute is responsible for easily initialize the pipelining preprocessing process.

Usage

```
pipeline_execute(configurationFilePath = NULL, editConfigurationFile = FALSE, filesPath,
pipe = SerialPipe$new(), instanceFactory = InstanceFactory$new())
```

Arguments

configurationFilePath (*character*) path where the configuration file is located. The file must have the .ini extension. In the case that the argument is null, the default configuration file (configurationsTemplate.ini) will be used.

editConfigurationFile (*boolean*) indicates whether a file is opened to modify the configuration file or not.

filesPath (*character*) path where the files to be preprocessed are located.

pipe (*TypePipe*) subclass of [TypePipe](#), which implements the whole pipelining process.

instanceFactory (*InstanceFactory*) object implementing the method `createInstance` to choose which type of [Instance](#) is created.

Value

List of [Instance](#) that have been preprocessed.

Details

The configuration file can be indicated by the user or use the default configuration file (configurationsTemplate.ini). In addition, once we call the function, it will be possible to choose if the user wants to edit the file of indicated configurations or not.

The *configurationFilePath* file should have the following structure (Depends on the Pipes used). Also the configurationsTemplate.ini has this structure:

[twitter]

ConsumerKey = «consumer_key»

ConsumerSecret = «consumer_secret»

AccessToken = «access_token»

AccessTokenSecret = «access_token_secret»

[youtube]app_id = «*app_id*»app_password = «*app_password*»**[eml]**PartSelectedOnMPAlternative = «*part_selected*» (text/html or text/plain)**[resourcesPath]**resourcesAbbreviationsPath = «*resources_abbreviations_path*»resourcesContractionsPath = «*resources_contractions_path*»resourcesInterjectionsPath = «*resources_interjections_path*»resourcesSlangsPath = «*resources_slangs_path*»resourcesStopWordsPath = «*resources_stopWords_path*»**[CSVPath]**outPutTeeCSVPipePath = «*out_put_teeCSVPipe_path*»**[cache]**cachePathTwid = «*cache_path_twid*»cachePathYtbid = «*cache_path_ytbid*»**Notes**

In the case of choosing to edit the configuration file, the default editor will be opened.

Examples

```
## Not run:
#Path where the configuration file are located
configurationFilePath <- system.file(file.path("examples",
                                             "configurationsExample.ini"),
                                     package = "bdpar")

#Folder with the files to preprocess
filesPath <- system.file(file.path("examples",
                                   "testFiles"),
                        package = "bdpar")

#Object which indicates the pipes' flow
pipe <- SerialPipe$new()

#Object which decides how creates the instances
instanceFactory <- InstanceFactory$new()

#Starting file preprocessing...
pipeline_execute(configurationFilePath = configurationFilePath,
                filesPath = filesPath,
                pipe = pipe,
                instanceFactory = instanceFactory)

## End(Not run)
```

ResourceHandler	<i>Class that handles different types of resources</i>
-----------------	--

Description

Class that handles different types of resources.

Usage

ResourceHandler

Constructor

ResourceHandler\$new()

Details

It is a class that allows store the resources that are needed in the Pipes to avoid having to repeatedly read from the file. File resources of type json are read and stored in memory.

Methods

- **isLoadResource:** from the resource path, it is checked if they have already been loaded. In this case, the list of the requested resource is returned. Otherwise, the resource variable is added to the list of resources, and the resource list is returned. In the event that the resource file does not exist, NULL is returned.
 - *Usage:* isLoadResource(pathResource)
 - *Arguments:*
 - * **pathResource:** (*character*) resource file path.
- **getResources:** gets of resources variable.
 - *Usage:* getResources()
 - *Value:* value of resources variable.
- **setResources:** sets of resources.
 - *Usage:* setResources(resources)
 - *Arguments:*
 - * **resources:** (*list*) the new value of resources.
- **getNamesResources:** gets of names of resources.
 - *Usage:* getNamesResources()
 - *Value:* value of names of resources.

Private fields

- **resources:** (*list*) variable that stores the lists of the different types of resources.

SerialPipe	<i>Class implementing a default pipelining process.</i>
------------	---

Description

This [SerialPipe](#) class inherits from the [TypePipe](#) class. Includes the **pipeAll** method which provides a default pipelining implementation.

Usage

```
SerialPipe
```

Constructor

```
SerialPipe$new()
```

Details

This class uses the default implementation provided by its parent class ([TypePipe](#)) and handles all the unexpected errors that may appear during the execution of the pipelining process.

Inherit

This class inherits from [TypePipe](#) and implements the `pipeAll` abstract function.

Methods

- **pipeAll**: implementation of the `pipeAll` function provided by the super class ([TypePipe](#)). In this case, the `pipeAll` function is called so that the default pipelining is executed.
 - *Usage*: `pipeAll(instance)`
 - *Value*: The preprocessed [Instance](#).
 - *Arguments*:
 - * **instance**: (*Instance*) The [Instance](#) that is going to be processed.

See Also

[Instance](#), [TypePipe](#)

SlangPipe

Class to find and/or replace the slangs on the data field of an Instance

Description

[SlangPipe](#) class is responsible for detecting the existing slangs in the **data** field of each [Instance](#). Identified slangs are stored inside the **slang** field of [Instance](#) class. Moreover if needed, is able to perform inline slangs replacement.

Usage

SlangPipe

Constructor

```
SlangPipe$new(propertyName = "langpropname",  
               propertyLanguageName = "language",  
               alwaysBeforeDeps = list("GuessLanguagePipe"),  
               notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **propertyLanguageName:** (*character*) name of the language property.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

[SlangPipe](#) class requires the resource files (in json format) containing the correspondence between slangs and meaning. To this end, the language of the text indicated in the *propertyLanguageName* should be contained in the resource file name (ie. slang.xxx.json where xxx is the value defined in the *propertyLanguageName*). The location of the resources should be defined in the *resourcesPath* section of the configuration file.

[resourcesPath]

```
resourcesSlangsPath = «resources_slangs_path»
```

Note

[SlangPipe](#) will automatically invalidate the [Instance](#) whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the [Instance](#) to obtain/replace the slangs. The slangs found in the Pipe are added to the list of properties of the [Instance](#) If the `replaceSlangs` parameter is `TRUE`, the [Instance](#) data will be modified by replacing the slangs found.
 - *Usage*: `pipe(instance, replaceSlangs = TRUE)`
 - *Value*: the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.
 - * **replaceSlangs**: (*logical*) indicate if the slangs are replace or not.
- **findSlang**: checks if the slang is in the data.
 - *Usage*: `findSlang(data, slang)`
 - *Value*: boolean, depending on whether the slang is on the data.
 - *Arguments*:
 - * **data**: (*character*) text where slang will be searched. **slang**: (*character*) indicates the slang to find.
- **replaceSlang**: replaces the slang in the data for the `extendedSlang`.
 - *Usage*:
`replaceSlang(slang, extendedSlang, data)`
 - *Value*: the data with slangs replaced.
 - *Arguments*:
 - * **slang**: (*character*) indicates the slang to replace.
 - * **extendedSlang**: (*character*) indicates the string to replace for the slangs found.
 - * **data**: (*character*) text where slang will be replaced.
- **getPropertyLanguageName**: gets of name of property language.
 - *Usage*:
`getPropertyLanguageName()`
 - *Value*: value of name of property language.
- **getResourcesSlangsPath**: gets of path of slangs resources.
 - *Usage*:
`getResourcesSlangsPath()`
 - *Value*:
value of path of slangs resources.
- **setResourcesSlangsPath**: sets the path of slangs resources.
 - *Usage*: `setResourcesSlangsPath(path)`
 - *Arguments*:
 - * **path**: (*character*) the new value of the path of slangs resources.

Private fields

- **propertyLanguageName**: (*character*) the name of property about language.
- **resourcesSlangsPath**: (*character*) the path where are the resources.

See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, PipeGeneric, ResourceHandler, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

StopWordPipe	<i>Class to find and/or remove the stop words on the data field of an Instance</i>
--------------	--

Description

`StopWordPipe` class is responsible for detecting the existing stop words in the **data** field of each `Instance`. Identified stop words are stored inside the **contraction** field of `Instance` class. Moreover if needed, is able to perform inline stop words removal.

Usage

```
StopWordPipe
```

Constructor

```
StopWordPipe$new(propertyName = "stopWord",
                  propertyLanguageName = "language",
                  alwaysBeforeDeps = list("GuessLanguagePipe"),
                  notAfterDeps = list("AbbreviationPipe"))
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **propertyLanguageName:** (*character*) name of the language property.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

`StopWordPipe` class requires the resource files (in json format) containing the list of stop words. To this end, the language of the text indicated in the *propertyLanguageName* should be contained in the resource file name (ie. xxx.json where xxx is the value defined in the *propertyLanguageName*). The location of the resources should be defined in the *resourcesPath* section of the configuration file.

[resourcesPath]

```
resourcesStopWordsPath = «resources_stopWords_path»
```

Note

`StopWordPipe` will automatically invalidate the `Instance` whenever the obtained data is empty.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain/remove the stop words. The stop words found in the pipe are added to the list of properties of the [Instance](#). If the *removeStopWords* parameter is TRUE, the [Instance](#) data will be removed.
 - *Usage:* `pipe(instance, removeStopWords = TRUE)`
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **removeStopWords:** (*logical*) indicates if the stop words are removed or not.
- **findStopWord:** checks if the stop word is in the data.
 - *Usage:* `findStopWord(data, stopWord)`
 - *Value:* boolean, depending on whether the stop word is on the data.
 - *Arguments:*
 - * **data:** (*character*) text where stop words will be searched.
 - * **stopWord:** (*character*) Indicates the stop word to find.
- **removeStopWord:** removes the stop word in the data.
 - *Usage:* `removeStopWord(stopWord, data)`
 - *Value:* the data with stop word removed.
 - *Arguments:*
 - * **stopWord:** (*character*) indicates the stop word to remove.
 - * **data:** (*character*) text where stop words will be removed.
- **getPropertyLanguageName:** gets of name of property language.
 - *Usage:* `getPropertyLanguageName()`
 - *Value:* value of name of property language.
- **getPathResourcesStopWords:** gets of path of stop words resources.
 - *Usage:* `getPathResourcesStopWords()`
 - *Value:* value of path of stop words resources.
- **setPathResourcesStopWords:** sets the path of stop words resources.
 - *Usage:* `setPathResourcesStopWords(path)`
 - *Arguments:*
 - * **path:** (*character*) the new value of the path of stop words resources.

Private fields

- **propertyLanguageName:** (*character*) the name of property about language.
- **pathResourcesStopWords:** (*character*) the path where are the resources.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

StoreFileExtPipe	<i>Class to get the file's extension field of an Instance</i>
------------------	---

Description

Gets the extension of a file. Creates the **extension** property which indicates extension of the file.

Usage

```
StoreFileExtPipe
```

Constructor

```
StoreFileExtPipe$new(propertyName = "extension",
                      alwaysBeforeDeps = list(),
                      notAfterDeps = list())
```

- *Arguments:*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Note

[StoreFileExtPipe](#) will automatically invalidate the [Instance](#) if it is not able to find the extension from the path field.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to obtain the extension of [Instance](#).
 - *Usage:* pipe(instance)
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.

- **obtainExtension:** gets of extension of the path.
 - *Usage:* obtainExtension(path)
 - *Value:* extension of the path.
 - *Arguments:*
 - * **path:** (*character*) path of the file to get the extension.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

TargetAssigningPipe *Class to get the target field of the Instance*

Description

This class allows searching in the path the **target** of the [Instance](#).

Usage

TargetAssigningPipe

Constructor

```
TargetAssigningPipe$new(targets = list("ham", "spam"),
                        targetsName = list("_ham_", "_spam_"),
                        propertyName = "target",
                        alwaysBeforeDeps = list(),
                        notAfterDeps = list())
```

- *Arguments:*
 - **targets:** (*list*) name of the targets property.
 - **targetsName:** (*list*) the name of folders.
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The targets that are searched can be controlled through the constructor of the class where *targetsName* will be the string that is searched within the path and *targets* has the values that the property can take.

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe**: preprocesses the [Instance](#) to obtain the target.
 - *Usage*: pipe(instance)
 - *Value*: The [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments*:
 - * **instance**: (*Instance*) [Instance](#) to preprocess.
- **getTarget** gets the target from a path.
 - *Usage*:
getTarget(path)
 - *Value*: the target of the path.
 - *Arguments*:
 - * **path**: (*character*) path to analyze.
- **checkTarget**: checks if the target is in the path.
 - *Usage*: checkTarget(target, path)
 - *Value*: if the target is found, returns target, else returns "".
 - *Arguments*:
 - * **target**: (*character*) target to find in the path.
 - * **path**: (*character*) path to analyze.
- **getTargets**: gets of targets.
 - *Usage*: getTargets()
 - *Value*: value of targets.

Private fields

- **targets**: (*list*) name of the targets property.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TeeCSVPipe](#), [ToLowerCasePipe](#)

TeeCSVPipe	<i>Class to handle a CSV with the properties field of the preprocessed Instance</i>
------------	---

Description

Complete a CSV with the properties of the preprocessed [Instance](#).

Usage

TeeCSVPipe

Constructor

```
TeeCSVPipe$new(propertyName = "",
                alwaysBeforeDeps = list(),
                notAfterDeps = list())
```

- *Arguments*

- **propertyName:** (*character*) name of the property associated with the Pipe.
- **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
- **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Details

The path to save the properties have to indicate in the configuration file.

The way to indicate is the following:

[CSVPath]

outPutTeeCSVPipePath = «out_put_TeeCSVPipe_path»

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** completes the CSV with the preprocessed [Instance](#).
 - *Usage:* pipe(instance,withData = TRUE,withSource = TRUE)
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
 - * **withData:** (*logical*) indicate if the data is added to CSV.
 - * **withSource:** (*logical*) indicate if the source is added to CSV.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [ToLowerCasePipe](#)

ToLowerCasePipe	<i>Class to convert the data field of an Instance to lower case</i>
-----------------	---

Description

Class to convert the data field of an [Instance](#) to lower case.

Usage

```
ToLowerCasePipe
```

Format

An object of class R6ClassGenerator of length 24.

Constructor

```
ToLowerCasePipe$new(propertyName = "",
                    alwaysBeforeDeps = list(),
                    notAfterDeps = list())
```

- *Arguments:*
 - **propertyName:** (*character*) name of the property associated with the Pipe.
 - **alwaysBeforeDeps:** (*list*) the dependences alwaysBefore (Pipes that must be executed before this one).
 - **notAfterDeps:** (*list*) the dependences notAfter (Pipes that cannot be executed after this one).

Inherit

This class inherits from [PipeGeneric](#) and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the [Instance](#) to convert the data to lower case.
 - *Usage:* pipe(instance)
 - *Value:* the [Instance](#) with the modifications that have occurred in the Pipe.
 - *Arguments:*
 - * **instance:** (*Instance*) [Instance](#) to preprocess.
- **toLowerCase:** converts the data to lower case.

- *Usage:* toLowerCase(data)
- *Value:* data in lower case.
- *Arguments:*
 - * **data:** (*character*) text to preprocess.

See Also

[AbbreviationPipe](#), [ContractionPipe](#), [File2Pipe](#), [FindEmojiPipe](#), [FindEmoticonPipe](#), [FindHashtagPipe](#), [FindUrlPipe](#), [FindUserNamePipe](#), [GuessDatePipe](#), [GuessLanguagePipe](#), [Instance](#), [InterjectionPipe](#), [MeasureLengthPipe](#), [PipeGeneric](#), [ResourceHandler](#), [SlangPipe](#), [StopWordPipe](#), [StoreFileExtPipe](#), [TargetAssigningPipe](#), [TeeCSVPipe](#)

TypePipe

Abstract super class implementing the pipelining process.

Description

Class to establish the flow of Pipes.

Usage

TypePipe

Format

An object of class R6ClassGenerator of length 24.

Constructor

TypePipe\$new()

Details

The default flow is:

instance %>I%

TargetAssigningPipe\$new()\$pipe() %>I%

StoreFileExtPipe\$new()\$pipe() %>I%

GuessDatePipe\$new()\$pipe() %>I%

File2Pipe\$new()\$pipe() %>I%

MeasureLengthPipe\$new()\$pipe("length_before_cleaning_text") %>I%

```

FindUserNamePipe$new()$pipe() %>I%
FindHashtagPipe$new()$pipe() %>I%
FindUrlPipe$new()$pipe() %>I%
FindEmoticonPipe$new()$pipe() %>I%
FindEmojiPipe$new()$pipe() %>I%
GuessLanguagePipe$new()$pipe() %>I%
ContractionPipe$new()$pipe() %>I%
AbbreviationPipe$new()$pipe() %>I%
SlangPipe$new()$pipe() %>I%
ToLowerCasePipe$new()$pipe() %>I%
InterjectionPipe$new()$pipe() %>I%
StopWordPipe$new()$pipe() %>I%
MeasureLengthPipe$new()$pipe("length_after_cleaning_text") %>I%
TeeCSVPipe$new()$pipe()

```

Methods

- **pipeAll**: function where is implemented the flow of the pipes.
 - *Usage*: pipeAll(instance)
 - *Value*: the preprocessed [Instance](#).
 - *Arguments*:
 - * **instance**: (*Instance*) the [Instance](#) that is going to be processed.

See Also

[Instance](#), [SerialPipe](#)

Description

Defines a customized forward pipe operator extending the features of classical %>%. Concretely %>I% is able to stop the pipelining process whenever an [Instance](#) has been invalidated. This issue, avoids executing the whole pipelining process for the invalidated [Instance](#) and therefore reduce the time and resources used to complete the whole process.

Usage

```
lhs %>I% rhs
```

Arguments

lhs	an Instance object.
rhs	a function call using the bdpars semantics.

Value

the [Instance](#) modified by the methods it has traversed.

Details

This is the %>% operator of the modified magrittr library in order to stop the flow when the [Instance](#) is invalid.

Note

Pipelining process is automatically stopped if the [Instance](#) is invalid.

See Also

[Instance](#)

Index

- [%>I%](#), 50
- [AbbreviationPipe](#), 2, 2, 3, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [Bdpar](#), 5
- [Connections](#), 6, 7, 7, 14, 15
- [ContractionPipe](#), 4, 9, 9, 10, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [ExtractorEml](#), 11, 13, 14, 16, 30
- [ExtractorSms](#), 12, 12, 14, 16, 30
- [ExtractorTwtid](#), 9, 12, 13, 13, 16, 30
- [ExtractorYtbid](#), 9, 12–14, 15, 30
- [File2Pipe](#), 4, 11, 16, 16, 18, 20, 21, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [FindEmojiPipe](#), 4, 11, 17, 17, 20, 21, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [FindEmoticonPipe](#), 4, 11, 17, 18, 18, 19, 21, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [FindHashtagPipe](#), 4, 11, 17, 18, 20, 20, 23, 25–27, 33–35, 42, 44–46, 48, 49
- [FindUrlPipe](#), 4, 11, 17, 18, 20, 21, 21, 22, 25–27, 33–35, 42, 44–46, 48, 49
- [FindUserNamePipe](#), 4, 11, 17, 18, 20, 21, 23, 24, 24, 26, 27, 33–35, 42, 44–46, 48, 49
- [GuessDatePipe](#), 4, 11, 17, 18, 20, 21, 23, 25, 25, 27, 33–35, 42, 44–46, 48, 49
- [GuessLanguagePipe](#), 4, 11, 17, 18, 20, 21, 23, 25, 26, 26, 33–35, 42, 44–46, 48, 49
- [Instance](#), 2–4, 6, 9–27, 27, 28–36, 39–51
- [InstanceFactory](#), 6, 30, 30
- [InterjectionPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 31, 31, 34, 35, 42, 44–46, 48, 49
- [MeasureLengthPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33, 33, 35, 42, 44–46, 48, 49
- [PipeGeneric](#), 3, 4, 10, 11, 16–27, 31, 33, 34, 34, 40, 42–49
- [pipeline_execute](#), 6, 36
- [ResourceHandler](#), 4, 11, 33–35, 38, 42, 44–46, 48, 49
- [SerialPipe](#), 6, 39, 39, 50
- [SlangPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 40, 40, 44–46, 48, 49
- [StopWordPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 42, 45, 46, 48, 49
- [StoreFileExtPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44, 44, 46, 48, 49
- [TargetAssigningPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44, 45, 45, 48, 49
- [TeeCSVPipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44–46, 47, 49
- [ToLowerCasePipe](#), 4, 11, 17, 18, 20, 21, 23, 25–27, 33–35, 42, 44–46, 48, 48
- [TypePipe](#), 5, 6, 36, 39, 49