Package 'bdpar'

February 20, 2020

Type Package

Title Big Data Preprocessing Architecture

Version 2.0.0

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 2020-02-20

License GPL-3

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

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

Depends R (>= 3.5.0)

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

Suggests cld2, knitr, readr, rex, rjson, rmarkdown, rtweet, stringi, stringr, testthat (>= 2.3.1), 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 2020-02-20 10:40:03 UTC

R topics documented:

`AbbreviationPipe` *Class to find and/or replace the abbreviations on the data field of an Instance*

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(),
                     replaceAbbreviations = TRUE,
                     resourcesAbbreviationsPath = NULL)
```

- *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).
  - **replaceAbbreviations:** (*logical*) indicates if the abbreviations are replaced or not.
  - **resourcesAbbreviationsPath:** (*character*) path of resource files (in json format) containing the correspondence between abbreviations and meaning.

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 **"resources.abbreviations.path"** field of *bdpar.Options* variable.

Note

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

Inherit

This class inherits from `GenericPipe` 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`.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.

- **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 abbreviatons 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*) path of resource files (in json format) containing the correspondence between abbreviations and meaning.
- **replaceAbbreviations:** (*logical*) indicates if the abbreviations are replaced or not.

See Also

`bdpar.Options`, `ContractionPipe`, `File2Pipe`, `FindEmojiPipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `FindUserNamePipe`, `GuessDatePipe`, `GuessLanguagePipe`, `Instance`, `InterjectionPipe`, `MeasureLengthPipe`, `GenericPipe`, `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) initialize the objects of handle the connections to APIs (Connections) and handles json resources (ResourceHandler) and (ii) executing the flow of pipes (inherited from GenericPipeline class) passed as argument.

Usage

```
Bdpar
```

Constructor

```
Bdpar$new()
```

Details

In the case that some pipe, defined on the workflow, needs some type of configuration, it can be defined throught *bdpar.Options* variable which have differents methods to support the funcionality of different pipes.

Static variables

- **connections**: (*Connections*) object that handles the connections with YouTube and Twitter.
- **resourceHandler**: (*ResourceHandler*) object that handles the json resources files.

Methods

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

    ```
    execute(path,
            extractors = ExtractorFactory$new(),
            pipeline = GenericPipeline$new())
    ```
  - *Value*: list of Instances that have been preprocessed.
  - *Arguments*:
    - **path**: (*character*) path where the files to be processed are located.
    - **extractors**: (*ExtractorFactory*) class which implements the createInstance method to choose which type of Instance is created.
    - **pipeline**: (*GenericPipeline*) subclass of GenericPipeline, which implements the execute method.

See Also

bdpar.Options, Connections, DefaultPipeline, DynamicPipeline, GenericPipeline,Instance, ExtractorFactory, ResourceHandler, runPipeline

Examples

```
## Not run:

#If it is necessary to indicate any existing configuration key, do it through:
#bdpar.Options$set(key, value)
#If the key is not initialized, do it through:
#bdpar.Options$add(key, value)

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

#Object which decides how creates the instances
extractors <- ExtractorFactory$new()

#Object which indicates the pipes' flow
pipeline <- DefaultPipeline$new()

objectBdpar <- Bdpar$new()

#Starting file preprocessing...
objectBdpar$execute(path = path,
                    extractors = extractors,
                    pipeline = pipeline)

## End(Not run)
```

`bdpar.Options` *Object to handle the keys/attributes/options common to all pipeline flow*

Description

This class provides the necessary methods to manage a list of keys or options used along the pipe flow, both those provided by the default library and those implemented by the user.

Usage

```
bdpar.Options
```

Details

By default, the application initializes the object named `bdpar.Options` of type `BdparOptions` which is in charge of initializing the options used in the defined pipes.

The default fields on *bdpar.Options* are initialized, if needed, as shown bellow:

[eml]

- `bdpar.Options$set("extractorEML.mpaPartSelected",<<PartSelectedOnMPAlternative>>)`

[resources]

- `bdpar.Options$set("resources.abbreviations.path",<<abbreviation.path>>)`
- `bdpar.Options$set("resources.contractions.path",<<contractions.path>>)`
- `bdpar.Options$set("resources.interjections.path",<<interjections.path>>)`
- `bdpar.Options$set("resources.slangs.path",<<slangs.path>>)`
- `bdpar.Options$set("resources.stopwords.path",<<stopwords.path>>)`

[twitter]

- `bdpar.Options$set("twitter.consumer.key",<<consumer_key>>)`
- `bdpar.Options$set("twitter.consumer.secret",<<consumer_secret>>)`
- `bdpar.Options$set("twitter.access.token",<<access_token>>)`
- `bdpar.Options$set("twitter.access.token.secret",<<access_token_secret>>)`
- `bdpar.Options$set("cache.twitter.path",<<cache.path>>)`

[teeCSVPipe]

- `bdpar.Options$set("teeCSVPipe.output.path",<<outputh.path>>)`

[youtube]

- `bdpar.Options$set("youtube.app.id",<<app_id>>)`
- `bdpar.Options$set("youtube.app.password",<<app_password>>)`
- `bdpar.Options$set("cache.youtube.path",<<cache.path>>)`

Methods

- **get:** obtains a specific option.
  - *Usage:* `get(key)`
  - *Value:* the value of the specific option.
  - *Arguments:*
    - **key:** (*character*) the name of the option to obtain.
- **add:** adds a option to the list of options
  - *Usage:* `add(key,value)`
  - *Arguments:*
    - **key:** (*character*) the name of the new option.
    - **propertyName:** (*Object*) the value of the new option.
- **set:** modifies the value of the one option.
  - *Usage:* `set(key,value)`

  - *Arguments:*
    * **key:** (*character*) the name of the new option.
    * **propertyName:** (*Object*) the value of the new option.
- **remove:** removes a specific option.
  - *Usage:* `remove(key)`
  - *Arguments:*
    * **key:** (*character*) the name of the option to remove.
- **getAll:** gets the list of options.
  - *Usage:* `getAll()`
  - *Value:* Value of options.
- **remove:** resets the option list to the initial state.
  - *Usage:* `reset()`
- **isSpecificOption:** checks for the existence of an specific option.
  - *Usage:* `isSpecificProperty(key)`
  - *Value:* A boolean results according to the existence of the specific option in the list of options
  - *Arguments:*
    * **key:** (*character*) the key of the option to check.

See Also

`AbbreviationPipe`, `Connections`, `ContractionPipe`, `ExtractorEml`, `ExtractorTwtid`, `ExtractorYtbid`, `GuessLanguagePipe`, `SlangPipe`, `StopWordPipe`, `TeeCSVPipe`

`bdparData` *Example of the content of the files to be preprocessed.*

Description

A manually collected data set containing e-mails and SMS messages from the nutritional and health domain classified as spam and non-spam (with a ratio of 50%). In addition the dataset contains two variables: (i) path which indicates the location of the target file and, (ii) source which contains the raw text comprising each file.

Usage

```
data(bdparData)
```

Format

A data frame with 20 rows and 2 variables:

path File path.

source File content.

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()
```

Details

The way to indicate the keys of YouTube and Twitter has to be through fields of *bdpar.Options* variable:

[twitter]

- `bdpar.Options$set("twitter.consumer.key",<<consumer_key>>)`
- `bdpar.Options$set("twitter.consumer.secret",<<consumer_secret>>)`
- `bdpar.Options$set("twitter.access.token",<<access_token>>)`
- `bdpar.Options$set("twitter.access.token.secret",<<access_token_secret>>)`

[youtube]

- `bdpar.Options$set("youtube.app.id",<<app_id>>)`
- `bdpar.Options$set("youtube.app.password",<<app_password>>)`

Note

Fiels 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 maximun 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

`bdpar.Options`, `ExtractorTwtid`, `ExtractorYtbid`

`ContractionPipe` *Class to find and/or replace the contractions on the data field of a Instance*

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

```
ContractionsPipe$new(propertyName = "contractions",
                     propertyLanguageName = "language",
                     alwaysBeforeDeps = list("GuessLanguagePipe"),
                     notAfterDeps = list(),
                     replaceContractions = TRUE,
                     resourcesContractionsPath = NULL)
```

- *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).
  - **replaceContractions:** (*logical*) indicates if the contractions are replace or not.
  - **resourcesContractionsPath:** (*character*) path of resource files (in json format) containing the correspondence between contractions and meaning.

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 **"resources.contractions.path"** field of *bdpar.Options* variable.

Note

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

Inherit

This class inherits from `GenericPipe` 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`.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **findContraction:** checks if the contractions 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 of 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*) path of resource files (in json format) containing the correspondence between contractions and meaning.
- **replaceContractions:** (*logical*) indicates if the contractions are replace or not.

See Also

`AbbreviationPipe`, `bdpar.Options`, `File2Pipe`, `FindEmojiPipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `FindUserNamePipe`, `GuessDatePipe`, `GuessLanguagePipe`, `Instance`, `InterjectionPipe`, `MeasureLengthPipe`, `GenericPipe`, `ResourceHandler`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TargetAssigningPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

DefaultPipeline *Class implementing a default pipelining proccess.*

Description

This DefaultPipeline class inherits from the GenericPipeline class. Includes the **execute** method which provides a default pipelining implementation.

Usage

```
DefaultPipeline
```

Constructor

```
DefaultPipeline$new()
```

Details

The default flow is:

```
instance %>|%

  TargetAssigningPipe$new() %>|%

  StoreFileExtPipe$new() %>|%

  GuessDatePipe$new() %>|%

  File2Pipe$new() %>|%

  MeasureLengthPipe$new(propertyName = "length_before_cleaning_text") %>|%

  FindUserNamePipe$new() %>|%

  FindHashtagPipe$new() %>|%

  FindUrlPipe$new() %>|%

  FindEmoticonPipe$new() %>|%

  FindEmojiPipe$new() %>|%

  GuessLanguagePipe$new() %>|%

  ContractionPipe$new() %>|%

  AbbreviationPipe$new() %>|%
```

```
SlangPipe$new() %>|%

ToLowerCasePipe$new() %>|%

InterjectionPipe$new() %>|%

StopWordPipe$new() %>|%

MeasureLengthPipe$new(propertyName = "length_after_cleaning_text") %>|%

TeeCSVPipe$new()
```

Inherit

This class inherits from GenericPipeline and implements the execute abstract function.

Methods

- **execute:** function where is implemented the flow of the pipes.
  - *Usage:* `execute(instance)`
  - *Value:* the preprocessed Instance.
  - *Arguments:*
    - **instance:** (*Instance*) the Instance that is going to be processed.
- **get:** gets a list with containinig the set of pipes of the pipeline,
  - *Usage:* `get()`
  - *Value:* the set of pipes containing the pipeline.

See Also

Instance, DynamicPipeline, GenericPipeline, GenericPipe, %>|%

`DynamicPipeline` *Class implementing a dynamic pipelining proccess.*

Description

This DynamicPipeline class inherits from the GenericPipeline class. Includes the **execute** method which provides a dynamic pipelining implementation.

Usage

```
DynamicPipeline
```

Constructor

```
DynamicPipeline$new(pipeline = NULL)
```

Inherit

This class inherits from `GenericPipeline` and implements the `execute` abstract function.

Methods

- **add:** adds a pipe or a pipe list to the pipeline
  - *Usage:* `add(pipe,pos)`
  - *Arguments:*
    - **pipe:** (*GenericPipe*) pipe objects or a list of pipes to add
    - **pos:** (*numeric*) the value of the pos to add. If it is NULL, pipe is appended to the pipeline
- **removeByPos:** removes pipes by the position on the pipeline
  - *Usage:* `removeByPos(pos)`
  - *Arguments:*
    - **pos:** (*numeric*) the pipe positions to remove.
- **removeByPipe:** removes pipes by its name on the pipeline
  - *Usage:* `removeByPipe(pipe.name)`
  - *Arguments:*
    - **pipe.name:** (*character*) the pipe name to remove.
- **removeAll:** removes all pipes included on pipeline
  - *Usage:* `removeAll()`
- **execute:** function where is implemented the flow of the pipes.
  - *Usage:* `execute(instance)`
  - *Value:* the preprocessed `Instance`.
  - *Arguments:*
    - **instance:** (*Instance*) the `Instance` that is going to be processed.
- **get:** gets a list with containinig the set of pipes of the pipeline,
  - *Usage:* `get()`
  - *Value:* the set of pipes containing the pipeline.

See Also

`Instance`, `DefaultPipeline`, `GenericPipeline`, `GenericPipe`, `%>|%`

`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.
  - **PartSelectedOnMPAlternative:** (*character*) configuration to read the eml files. If it is NULL, checks if is defined in the **"extractorEML.mpaPartSelected"** field of *bdpar.Options* variable.

Details

The way to indicate which part to choose in the email, when is a multipart email, is through the **"extractorEML.mpaPartSelected"** field of *bdpar.Options* variable.

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

`bdpar.Options`, `ExtractorSms`, `ExtractorTwtid`, `ExtractorYtbid`, `Instance`

`ExtractorFactory` *Class to handle the creation of Instance types*

Description

`ExtractorFactory` class builds the appropriate `Instance` object according to the file extension.

Usage

```
ExtractorFactory
```

Constructor

```
ExtractorFactory$new()
```

Methods

#'

registerExtractor: adds an extractor to the list of extensions

- *Usage:* `registerExtractor(extension,extractor)`
- *Arguments:*
  - **extension:** (*character*) the name of the extension option.
  - **extractor:** (*Object*) the extractor of the new extension.

- **setExtractor:** modifies the extractor of the one extension.
  - *Usage:* `setExtractor(extension,extractor)`
  - *Arguments:*

        * **extension:** (*character*) the name of the new extension.
        * **extractor:** (*Instance*) the value of the new extractor.
- **removeExtractor:** removes a specific extractor throught the extension.
  - *Usage:* `removeExtractor(extension)`
  - *Arguments:*
    * **extension:** (*character*) the name of the extension to remove.
- **getAllExtractors:** gets the list of extractors.
  - *Usage:* `getAllExtractors()`
  - *Value:* Value of extractors.
- **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`

`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,cachePath = NULL)
```

- *Arguments:*
  - **path:** (*character*) path of the twtid type file.
  - **cachePath:** (*character*) path of the cache location. If it is NULL, checks if is defined in the **"cache.twitter.path"** field of *bdpar.Options* variable.

Details

Twitter connection is handled through the `Connections` class which loads the Twitter API credentials from the *bdpar.Options* object. 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 **"cache.twitter.path"** field of *bdpar.Options* variable. This variable has to be the path to store the tweets and it is neccesary that it has two folder named: "_spam_" and "_ham_"

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

`bdpar.Options`, `Connections`, `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,cachePath = NULL)
```

- *Arguments:*
  - **path:** (*character*) path of the ytbid type file.
  - **cachePath:** (*character*) path of the cache location. If it is NULL, checks if is defined in the **"cache.youtube.path"** field of *bdpar.Options* variable.

Details

YouTube conection is handled through the `Connections` class which loads the YouTube API credentials from the *bdpar.Options* object. 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 **"cache.youtube.path"** field of *bdpar.Options* variable. This variable has to be the path to store the comments and it is neccesary that it has two folder named: "_spam_" and "_ham_"

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 perfomed 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

`bdpar.Options`, `Connections`, `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 GenericPipe 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 preproccess.

See Also

AbbreviationPipe, ContractionPipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

FindEmojiPipe *Class to find and/or replace the emoji on the data field of an Instance*

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(),
                  replaceEmojis = TRUE)
```

- *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).
  - **replaceEmojis:** (*logical*) indicates if the emojis are replaced.

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 GenericPipe and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the Instance to obtain/replace the emojis.
  - *Usage:* `pipe(instance)`
  - *Value:* the Instance with the modifications that have occurred in the Pipe.
  - *Arguments:*

        * **instance:** (*Instance*) `Instance` to preproccess.
    - **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.

Private fields

- **replaceEmojis:** (*logical*) indicates if the emojis are replaced.

See Also

`AbbreviationPipe`, `ContractionPipe`, `File2Pipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `FindUserNamePipe`, `GuessDatePipe`, `GuessLanguagePipe`, `Instance`, `InterjectionPipe`, `MeasureLengthPipe`, `GenericPipe`, `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 removement.

Usage

```
FindEmoticonPipe
```

Constructor

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

- *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).
  - **removeEmoticons:** (*logical*) indicates if the emoticons are replaced.

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 `GenericPipe` and implements the `pipe` abstract function.

Methods

- **pipe:** preprocesses the `Instance` to obtain/remove the emoticons.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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.

Private fields

- **removeEmoticons:** (*logical*) indicates if the emoticons are replaced.

See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

`FindHashtagPipe` *Class to find and/or remove the hashtags on the data field of an Instance*

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 removement.

Usage

```
FindHashtagPipe
```

Constructor

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

- *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).
  - **removeHashtag:** (*logical*) indicates if the hashstags are removed.

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 GenericPipe and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the Instance to obtain/remove the hashtags.
  - *Usage:* `pipe(instance)`
  - *Value* the Instance with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) Instance to preproccess.
- **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.

Private fields

- **removeHashtags:** (*logical*) indicates if the hashstags are removed.

See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, 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 removement.

Usage

```
FindUrlPipe
```

Constructor

```
FindUrlPipe$new(propertyName = "URLs",
                alwaysBeforeDeps = list(),
                notAfterDeps = list(),
                removeUrls = TRUE,
                URLPatterns = list(self$URLPattern, self$EmailPattern),
                namesURLPatterns = list("UrlPattern","EmailPattern"))
```

- *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).
  - **removeUrls:** (*logical*) indicates if the URLs are removed.
  - **URLPatterns:** (*list*) the regex to find URLs.
  - **namesURLPatterns:** (*list*) the names of regex.

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 GenericPipe and implements the pipe abstract function.

Methods

- **pipe:** preprocesses the `Instance` to obtain/remove the users.
  - *Usage:*
    ```
    pipe(instance)
    ```
  - *Value:*
    the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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.
- **removeUrls:** (*logical*) indicates if the URLs are removed.

See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

`FindUserNamePipe` *Class to find and/or remove the users on the data field of an Instance*

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 nanme removement.

Usage

```
FindUserNamePipe
```

Constructor

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

- *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).
  - **removeUser:** (*logical*) indicates if the users are removed.

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 `GenericPipe` and implements the `pipe` abstract function.

Methods

- **pipe:** preprocesses the `Instance` to obtain/remove the name users.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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.

Private fields

- **removeUser:** (*logical*) indicates if the users are removed.

See Also

`AbbreviationPipe`, `ContractionPipe`, `File2Pipe`, `FindEmojiPipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `GuessDatePipe`, `GuessLanguagePipe`, `Instance`, `InterjectionPipe`, `MeasureLengthPipe`, `GenericPipe`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TargetAssigningPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

`GenericPipe` *Abstract super classs that handles the management of the Pipes*

Description

Provides the required methods to succesfully handle each `GenericPipe` class.

Usage

```
GenericPipe
```

Constructor

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

- *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).

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

---

GenericPipeline *Absctract super class implementing the pipelining proccess.*

Description

Abstract super class to establish the flow of Pipes.

Usage

```
GenericPipeline
```

Constructor

```
GenericPipeline$new()
```

Methods

- **execute:** function where is implemented the flow of the pipes.
  - *Usage:* `execute(instance)`
  - *Value:* the preprocessed `Instance`.
  - *Arguments:*
    - **instance:** (*Instance*) the `Instance` that is going to be processed.
- **get:** gets a list with containinig the set of pipes of the pipeline,
  - *Usage:* `get()`
  - *Value:* the set of pipes containing the pipeline.

See Also

`DefaultPipeline`, `DynamicPipeline`, `Instance`, `GenericPipe`, `%>|%`

`GuessDatePipe` *Class to obtain the date field of an Instance*

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 `GenericPipe` 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 preproccess.

See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

GuessLanguagePipe *Class to guess the language of an Instance*

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(),
                      languageTwitter = TRUE)
```

- *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).
  - **languageTwitter:** (*logical*) indicates whether for the Instances of type twtid the language that returns the api is obtained or the detector is applied.

Details

To obtain the language of the tweets, it will be verified that there is a json file with the information stored in memory. On the other hand, it is necessary define the **"cache.twitter.path"** field of *bdpar.Options* variable to know where the information of tweets are saved.

Note

The Pipe will invalidate the `Instance` if the language of the data can not be detect.

Inherit

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

Methods

- **pipe:** preprocesses the `Instance` to obtain the language of the data.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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.

Private fields

- **languageTwitter:** (*logical*) indicates whether for the Instances of type twtid the language that returns the api is obtained or the detector is applied.

See Also

`AbbreviationPipe`, `bdpar.Options`, `ContractionPipe`, `File2Pipe`, `FindEmojiPipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `FindUserNamePipe`, `GuessDatePipe`, `Instance`, `InterjectionPipe`, `MeasureLengthPipe`, `GenericPipe`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TargetAssigningPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

`Instance` *Abstract super class that handles the management of the Instances*

Description

Provides the required methods to succesfully handle each `Instance` class.

Usage

```
Instance
```

Constructor

```
Instance$new(path)
```

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

Methods

- **obtainDate:** abstract function responsible for obtainining 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 contaning 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.

`InterjectionPipe` *Class to find and/or remove the interjections on the data field of an Instance*

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 removement.

Usage

```
InterjectionPipe
```

Constructor

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

- *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).
  - **removeInterjections:** (*logical*) indicates if the interjections are removed or not.

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 **"resources.interjections.path"** field of *bdpar.Options* variable.

Note

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

Inherit

This class inherits from `GenericPipe` 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`.
  - *Usage:*
    `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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.
- **removeInterjections:** (*logical*) indicates if the interjections are removed or not.

See Also

`AbbreviationPipe`, `bdpar.Options`, `ContractionPipe`, `File2Pipe`, `FindEmojiPipe`, `FindEmoticonPipe`, `FindHashtagPipe`, `FindUrlPipe`, `FindUserNamePipe`, `GuessDatePipe`, `GuessLanguagePipe`, `Instance`, `MeasureLengthPipe`, `GenericPipe`, `ResourceHandler`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TargetAssigningPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

`MeasureLengthPipe` *Class to obtain the length of the data field of an Instance*

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 throught the class constructor.

Usage

```
MeasureLengthPipe
```

Constructor

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

- *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).
  - **nchar_conf** (*logical*) indicates if the Pipe uses nchar or object.size.

Inherit

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

Methods

- **pipe:** preprocesses the `Instance` to obtain the length of data.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.
- **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 preproccess.
    - **nchar_conf:** (*logical*) indicates if the Pipe uses nchar or object.size.

Private fields

- **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`, `GenericPipe`, `ResourceHandler`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TargetAssigningPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

operator-pipe *bdpar customized fordward-pipe operator*

Description

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

Usage

```
lhs %>|% rhs
```

Arguments

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

Value

The Instance modified by the methods it has traversed.

Details

This is the %>% operator of the modified magrittr library to both (i) to stop the flow when the Instance is invalid and (ii) automatically call the pipe function of the R6 objects passing through it and (iii) check the dependencies of the Instance.

The usage structure would be as shown below:

```
instance %>|%

pipeObject$new() %>|%

pipeObject$new(<<argument1>>, <<argument2>, ...) %>|%

pipeObject$new()
```

Note

Pipelining proccess is automatically stopped if the Instance is invalid.

See Also

Instance, GenericPipe

`ResourceHandler` *Class that handles different types of resources*

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.

runPipeline *Initiates the pipelining process*

Description

runPipeline is responsible for easily initialize the pipelining preprocessing proccess.

Usage

```
runPipeline(path, extractors = ExtractorFactory$new(),
pipeline = DefaultPipeline$new())
```

Arguments

path	(*character*) path where the files to be preprocessed are located.
extractors	(*ExtractorFactory*) object implementing the method createInstance to choose which type of Instance is created.
pipeline	(*GenericPipeline*) subclass of GenericPipeline, which implements the whole pipeling process.

Value

List of Instance that have been preprocessed.

Details

In the case that some pipe, defined on the workflow, needs some type of configuration, it can be defined throught *bdpar.Options* variable which have differents methods to support the funcionality of different pipes.

See Also

Bdpar, bdpar.Options, Connections,DefaultPipeline, DynamicPipeline, GenericPipeline, Instance, ExtractorFactory, ResourceHandler

Examples

```
## Not run:

#If it is necessary to indicate any existing configuration key, do it through:
#bdpar.Options$set(key, value)
#If the key is not initialized, do it through:
#bdpar.Options$add(key, value)

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

```
#Object which decides how creates the instances
extractors <- ExtractorFactory$new()

#Object which indicates the pipes' flow
pipeline <- DefaultPipeline$new()

#Starting file preprocessing...
runPipeline(path = path,
            extractors = extractors,
            pipeline = pipeline)

## End(Not run)
```

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(),
              replaceSlangs = TRUE)
```

- *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).
  - **replaceSlangs:** (*logical*) indicates if the slangs are replace or not.

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 **"resources.slangs.path"** field of *bdpar.Options* variable.

Note

SlangPipe will automatically invalidate the Instance whenever the obtained data is empty.

Inherit

This class inherits from GenericPipe 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.
  - *Usage:* `pipe(instance)`
  - *Value:* the Instance with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) Instance to preproccess.
- **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.
- **replaceSlangs:** (*logical*) indicates if the slangs are replace or not.

See Also

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

---

StopWordPipe — *Class to find and/or remove the stop words on the data field of an Instance*

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 removement.

Usage

```
StopWordPipe
```

Constructor

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

- *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).
  - **removeStopWords:** (*logical*) indicates if the stop words are removed or not.

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 **"resources.stopwords.path"** field of *bdpar.Options* variable.

Note

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

Inherit

This class inherits from GenericPipe 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.
  - *Usage:* `pipe(instance)`
  - *Value:* the Instance with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) Instance to preproccess.
- **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.
- **removeStopWords:** (*logical*) indicates if the stop words are removed or not.

See Also

AbbreviationPipe, bdpar.Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

StoreFileExtPipe *Class to get the file's extension field of an Instance*

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 GenericPipe 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, GenericPipe, 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 `GenericPipe` 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 preproccess.
- **getTarget** gets the target from a path.
  - *Usage:*
    `getTarget(path)`
  - *Value:* the target of the path.
  - *Arguments:*
    - **path:** (*character*) path to analize.
- **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 analize.
- **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`, `GenericPipe`, `ResourceHandler`, `SlangPipe`, `StopWordPipe`, `StoreFileExtPipe`, `TeeCSVPipe`, `ToLowerCasePipe`

`TeeCSVPipe` *Class to handle a CSV with the properties field of the preprocessed Instance*

Description

Complete a CSV with the properties of the preprocessed `Instance`.

Usage

```
TeeCSVPipe
```

Constructor

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

- *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).
  - **withData:** (*logical*) indicates if the data is added to CSV.
  - **withSource:** (*logical*) indicates if the source is added to CSV.

Details

The path to save the properties should be defined in the **"teeCSVPipe.output.path"** field of *bdpar.Options* variable.

Inherit

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

Methods

- **pipe:** completes the CSV with the preprocessed `Instance`.
  - *Usage:* `pipe(instance)`
  - *Value:* the `Instance` with the modifications that have occurred in the Pipe.
  - *Arguments:*
    - **instance:** (*Instance*) `Instance` to preproccess.

Private fields

- **withSource:** (*logical*) indicates if the source is added to CSV.
- **withData:** (*logical*) indicates if the data is added to CSV.

See Also

AbbreviationPipe, bdpar.Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, ToLowerCasePipe

---

ToLowerCasePipe *Class to convert the data field of an Instance to lower case*

Description

Class to convert the data field of an Instance to lower case.

Usage

```
ToLowerCasePipe
```

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 GenericPipe 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 preproccess.

- **toLowerCase:** converts the data to lower case.
  - *Usage:* `toLowerCase(data)`
  - *Value:* data in lower case.
  - *Arguments:*
    - **data:** (*character*) text to preproccess.

See Also

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

Index