Package: wordpredictor (via r-universe)

December 8, 2024

Title Develop Text Prediction Models Based on N-Grams

Version 0.0.5

URL https://github.com/pakjiddat/word-predictor,
https://pakjiddat.github.io/word-predictor/

BugReports https://github.com/pakjiddat/word-predictor/issues

Description A framework for developing n-gram models for text prediction. It provides data cleaning, data sampling, extracting tokens from text, model generation, model evaluation and word prediction. For information on how n-gram models work we referred to: ``Speech and Language Processing" <https://web.archive.org/web/20240919222934/https%3A%2F%2Fweb.stanford.edu%2F~jurafsky%2Fslp3%2F3.pdf>. For optimizing R code and using R6 classes we referred to ``Advanced R" <https://adv-r.hadley.nz/r6.html>. For writing R extensions we referred to ``R Packages", <https://r-pkgs.org/index.html>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports digest, ggplot2, patchwork, stringr, dplyr, SnowballC

Suggests testthat, covr, knitr, rmarkdown, markdown

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Nadir Latif [aut, cre]
(<https://orcid.org/0000-0002-7543-7405>)

Maintainer Nadir Latif <pakjiddat@gmail.com>

Repository CRAN

Date/Publication 2024-10-08 16:20:02 UTC

Config/pak/sysreqs libicu-dev

Contents

`Base` *Base class for all other classes*

Description

Provides a basic structure for processing text files. Also provides methods for reading and writing files and objects.

Details

It provides pre-processing, processing and post-processing methods, which need to be overridden by derived classes.

The pre-processing function is called before reading a file. The process function is called for processing a given number of lines. The post processing function is called on the processed data.

Also provides methods for reading and writing text files and R objects. All class methods are private.

Methods

Public methods:

- `Base$new()`
- `Base$clone()`

Method `new()`**:** It initializes the current object. It is used to set the file name and verbose options.

Usage:

```
Base$new(fn = NULL, lc = 100, ve = 2)
```

Arguments:

`fn` The path to the file to clean.

`lc` The number of lines to read and clean at a time.

ve The level of detail in the information messages. Reads the given file one line at a time. It runs the given pre-processing function before reading the file. It runs the given

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Base$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`DataAnalyzer` *Analyzes input text files and n-gram token files*

Description

It provides a method that returns information about text files, such as number of lines and number of words. It also provides a method that displays bar plots of n-gram frequencies. Additionally it provides a method for searching for n-grams in a n-gram token file. This file is generated using the TokenGenerator class.

Details

It provides a method that returns text file information. The text file information includes total number of lines, max, min and mean line length and file size.

It also provides a method that generates a bar plot showing the most common n-gram tokens.

Another method is provided which returns a list of n-grams that match the given regular expression.

Super class

`wordpredictor::Base` -> `DataAnalyzer`

Methods

Public methods:

- `DataAnalyzer$new()`
- `DataAnalyzer$plot_n_gram_stats()`
- `DataAnalyzer$get_file_info()`
- `DataAnalyzer$get_ngrams()`
- `DataAnalyzer$clone()`

Method `new()`**:** It initializes the current object. It is used to set the file name and verbose options.

Usage:

```
DataAnalyzer$new(fn = NULL, ve = 0)
```

Arguments:

fn The path to the input file.

ve The level of detail in the information messages.

Method plot_n_gram_stats(): It allows generating two type of n-gram plots. It first reads n-gram token frequencies from an input text file. The n-gram frequencies are displayed in a bar plot.

The type of plot is specified by the type option. The type options can have the values 'top_features' or 'coverage'. 'top_features' displays the top n most occurring tokens along with their frequencies. 'coverage' displays the number of words along with their frequencies.

The plot stats are returned as a data frame.

Usage:

```
DataAnalyzer$plot_n_gram_stats(opts)
```

Arguments:

opts The options for analyzing the data.

- **type**. The type of plot to display. The options are: 'top_features', 'coverage'.
- **n**. For 'top_features', it is the number of top most occurring tokens. For 'coverage' it is the first n frequencies.
- **save_to**. The graphics devices to save the plot to. NULL implies plot is printed.
- **dir**. The output directory where the plot will be saved.

Returns: A data frame containing the stats.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL value implies tempdir will
# be used.
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n2.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram file name
nfn <- paste0(ed, "/n2.RDS")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(nfn, ve = ve)
# The top features plot is checked
df <- da$plot_n_gram_stats(opts = list(
    "type" = "top_features",
    "n" = 10,
    "save_to" = NULL,
    "dir" = ed
```

```
))
# N-gram statistics are displayed
print(df)
# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `get_file_info()`**:** It generates information about text files. It takes as input a file or a directory containing text files. For each file it calculates the total number of lines, maximum, minimum and mean line lengths and the total file size. The file information is returned as a data frame.

Usage:

```
DataAnalyzer$get_file_info(res)
```

Arguments:

`res` The name of a directory or a file name.

Returns: A data frame containing the text file statistics.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The test file name
cfn <- paste0(ed, "/test.txt")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(ve = ve)
# The file info is fetched
fi <- da$get_file_info(cfn)
# The file information is printed
print(fi)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `get_ngrams()`**:** It extracts a given number of n-grams and their frequencies from a n-gram token file.

The prefix parameter specifies the regular expression for matching n-grams. If this parameter is not specified then the given number of n-grams are randomly chosen.

Usage:

```
DataAnalyzer$get_ngrams(fn, c = NULL, pre = NULL)
```

Arguments:

`fn` The n-gram file name.

`c` The number of n-grams to return.

`pre` The n-gram prefix, given as a regular expression.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n2.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram file name
nfn <- paste0(ed, "/n2.RDS")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(nfn, ve = ve)
# Bi-grams starting with "and_" are returned
df <- da$get_ngrams(fn = nfn, c = 10, pre = "^and_*")
# The data frame is sorted by frequency
df <- df[order(df$freq, decreasing = TRUE),]
# The data frame is printed
print(df)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataAnalyzer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `DataAnalyzer$plot_n_gram_stats`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL value implies tempdir will
# be used.
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n2.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram file name
nfn <- paste0(ed, "/n2.RDS")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(nfn, ve = ve)
# The top features plot is checked
df <- da$plot_n_gram_stats(opts = list(
    "type" = "top_features",
    "n" = 10,
    "save_to" = NULL,
    "dir" = ed
))
# N-gram statistics are displayed
print(df)
# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `DataAnalyzer$get_file_info`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test.txt")
# An object of class EnvManager is created
```

```
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The test file name
cfn <- paste0(ed, "/test.txt")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(ve = ve)
# The file info is fetched
fi <- da$get_file_info(cfn)
# The file information is printed
print(fi)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `DataAnalyzer$get_ngrams`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n2.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram file name
nfn <- paste0(ed, "/n2.RDS")
# The DataAnalyzer object is created
da <- DataAnalyzer$new(nfn, ve = ve)
# Bi-grams starting with "and_" are returned
df <- da$get_ngrams(fn = nfn, c = 10, pre = "^and_*")
# The data frame is sorted by frequency
df <- df[order(df$freq, decreasing = TRUE),]
# The data frame is printed
print(df)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

DataCleaner *Provides data cleaning functionality*

Description

It provides a memory efficient method for removing unneeded characters from text files. It is suitable for cleaning large text files.

Details

It provides a method for cleaning text files. It allows removing bad words, stop words, non dictionary words, extra space, punctuation and non-alphabet characters. It also allows conversion to lower case. It supports large text files.

Super class

`wordpredictor::Base` -> `DataCleaner`

Methods

Public methods:

- `DataCleaner$new()`
- `DataCleaner$clean_file()`
- `DataCleaner$clean_lines()`
- `DataCleaner$clone()`

Method `new()`**:** It initializes the current object. It is used to set the file name and verbose options.

Usage:

```
DataCleaner$new(fn = NULL, opts = list(), ve = 0)
```

Arguments:

`fn` The path to the file to clean.

`opts` The options for data cleaning.

- **min_words**. The minimum number of words per sentence.
- **line_count**. The number of lines to read and clean at a time.
- **save_data**. If the combined processed lines should be saved.
- **output_file**. Name of the output file used to store the data.
- **sw_file**. The stop words file path.
- **dict_file**. The dictionary file path.
- **bad_file**. The bad words file path.
- **to_lower**. If the words should be converted to lower case.
- **remove_stop**. If stop words should be removed.
- **remove_punct**. If punctuation symbols should be removed.
- **remove_non_dict**. If non dictionary words should be removed.

- **remove_non_alpha**. -> If non alphabet symbols should be removed.
- **remove_extra_space**. -> If leading, trailing and double spaces should be removed.
- **remove_bad**. If bad words should be removed

`ve` The level of detail in the information messages.

Method `clean_file()`**:** It provides an efficient method for cleaning text files. It removes unneeded characters from the given text file with several options.

It allows removing punctuation, bad words, stop words, non-alphabetical symbols and non-dictionary words. It reads a certain number of lines from the given text file. It removes unneeded characters from the lines and then saves the lines to an output text file.

File cleaning progress is displayed if the verbose option was set in the class constructor. It is suitable for cleaning large text files.

Usage:

```
DataCleaner$clean_file()
```

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The cleaned test file name
cfn <- paste0(ed, "/test-clean.txt")
# The test file name
fn <- paste0(ed, "/test.txt")
# The data cleaning options
dc_opts <- list("output_file" = cfn)
# The data cleaner object is created
dc <- DataCleaner$new(fn, dc_opts, ve = ve)
# The sample file is cleaned
dc$clean_file()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clean_lines()`**:** It cleans the given lines of text using the options passed to the current object.

Usage:

```
DataCleaner$clean_lines(lines)
```

Arguments:

`lines` The input sentences.

Returns: The cleaned lines of text.

Examples:

```
# The level of detail in the information messages
ve <- 0
# Test data is read
l <- c(
    "If you think I'm wrong, send me a link to where it's happened",
    "We're about 90percent done with this room",
    "This isn't how I wanted it between us.",
    "Almost any cute breed can become ornamental",
    "Once upon a time there was a kingdom with a castle",
    "That's not a thing any of us are granted'",
    "Why are you being so difficult? she asks."
)
# The expected results
res <- c(
    "if you think wrong send me a link to where its happened",
    "were about percent done with this room",
    "this how i wanted it between us",
    "almost any cute breed can become ornamental",
    "once upon a time there was a kingdom with a castle",
    "thats not a thing any of us are granted",
    "why are you being so difficult she asks"
)
# The DataCleaner object is created
dc <- DataCleaner$new(ve = ve)
# The line is cleaned
cl <- dc$clean_lines(l)
# The cleaned lines are printed
print(cl)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataCleaner$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `DataCleaner$clean_file`
## ------------------------------------------------
```

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The cleaned test file name
cfn <- paste0(ed, "/test-clean.txt")
# The test file name
fn <- paste0(ed, "/test.txt")
# The data cleaning options
dc_opts <- list("output_file" = cfn)
# The data cleaner object is created
dc <- DataCleaner$new(fn, dc_opts, ve = ve)
# The sample file is cleaned
dc$clean_file()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `DataCleaner$clean_lines`
## ------------------------------------------------

# The level of detail in the information messages
ve <- 0
# Test data is read
l <- c(
    "If you think I'm wrong, send me a link to where it's happened",
    "We're about 90percent done with this room",
    "This isn't how I wanted it between us.",
    "Almost any cute breed can become ornamental",
    "Once upon a time there was a kingdom with a castle",
    "That's not a thing any of us are granted'",
    "Why are you being so difficult? she asks."
)
# The expected results
res <- c(
    "if you think wrong send me a link to where its happened",
    "were about percent done with this room",
    "this how i wanted it between us",
    "almost any cute breed can become ornamental",
    "once upon a time there was a kingdom with a castle",
```

```
    "thats not a thing any of us are granted",
    "why are you being so difficult she asks"
)
# The DataCleaner object is created
dc <- DataCleaner$new(ve = ve)
# The line is cleaned
cl <- dc$clean_lines(l)
# The cleaned lines are printed
print(cl)
```

DataSampler *Generates data samples from text files*

Description

It provides a method for generating training, testing and validation data sets from a given input text file.

It also provides a method for generating a sample file of given size or number of lines from an input text file. The contents of the sample file may be cleaned or randomized.

Super class

wordpredictor::Base -> DataSampler

Methods

Public methods:

- DataSampler$new()
- DataSampler$generate_sample()
- DataSampler$generate_data()
- DataSampler$clone()

Method `new()`**:** It initializes the current object. It is used to set the verbose option.

Usage:

```
DataSampler$new(dir = ".", ve = 0)
```

Arguments:

`dir` The directory for storing the input and output files.

`ve` The level of detail in the information messages.

Method `generate_sample()`**:** Generates a sample file of given size from the given input file. The file is saved to the directory given by the dir object attribute. Once the file has been generated, its contents may be cleaned or randomized.

Usage:

```
DataSampler$generate_sample(fn, ss, ic, ir, ofn, is, dc_opts = NULL)
```

Arguments:

`fn` The input file name. It is the short file name relative to the dir attribute.

`ss` The number of lines or proportion of lines to sample.

`ic` If the sample file should be cleaned.

`ir` If the sample file contents should be randomized.

`ofn` The output file name. It will be saved to the dir.

`is` If the sampled data should be saved to a file.

`dc_opts` The options for cleaning the data.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The sample file name
sfn <- paste0(ed, "/sample.txt")
# An object of class DataSampler is created
ds <- DataSampler$new(dir = ed, ve = ve)
# The sample file is generated
ds$generate_sample(
    fn = "input.txt",
    ss = 0.5,
    ic = FALSE,
    ir = FALSE,
    ofn = "sample.txt",
    is = TRUE
)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `generate_data()`**:** It generates training, testing and validation data sets from the given input file. It first reads the file given as a parameter to the current object. It partitions the data into training, testing and validation sets, according to the perc parameter. The files are named train.txt, test.txt and va.txt and are saved to the given output folder.

Usage:

```
DataSampler$generate_data(fn, percs)
```

Arguments:

`fn` The input file name. It should be relative to the dir attribute.

`percs` The size of the training, testing and validation sets.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be
# used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve)
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The files to clean
fns <- c("train", "test", "validate")
# An object of class DataSampler is created
ds <- DataSampler$new(dir = ed, ve = ve)
# The train, test and validation files are generated
ds$generate_data(
    fn = "input.txt",
    percs = list(
        "train" = 0.8,
        "test" = 0.1,
        "validate" = 0.1
    )
)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataSampler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
```

```
## Method `DataSampler$generate_sample`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The sample file name
sfn <- paste0(ed, "/sample.txt")
# An object of class DataSampler is created
ds <- DataSampler$new(dir = ed, ve = ve)
# The sample file is generated
ds$generate_sample(
    fn = "input.txt",
    ss = 0.5,
    ic = FALSE,
    ir = FALSE,
    ofn = "sample.txt",
    is = TRUE
)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `DataSampler$generate_data`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be
# used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve)
# The required files are downloaded
```

```
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The files to clean
fns <- c("train", "test", "validate")
# An object of class DataSampler is created
ds <- DataSampler$new(dir = ed, ve = ve)
# The train, test and validation files are generated
ds$generate_data(
    fn = "input.txt",
    percs = list(
        "train" = 0.8,
        "test" = 0.1,
        "validate" = 0.1
    )
)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

EnvManager *Allows managing the test environment*

Description

This class provides a method for creating directories in the tempdir folder for testing purposes. It also provides a method for reading files from the inst/extdata folder.

Super class

wordpredictor::Base -> EnvManager

Methods

Public methods:

- EnvManager$new()
- EnvManager$get_data_fn()
- EnvManager$remove_files()
- EnvManager$td_env()
- EnvManager$cp_env()
- EnvManager$setup_env()
- EnvManager$clone()

Method new(): It initializes the current object. It simply calls the base class constructor.

Usage:

```
EnvManager$new(rp = "../../", ve = 0)
```

Arguments:

`rp` The prefix for accessing the package root folder.

`ve` The level of detail in the information messages.

Method `get_data_fn()`**:** Checks if the given file exists. If it does not exist, then it tries to load the file from the inst/extdata data folder of the package. It throws an error if the file was not found. If the file exists, then the method simply returns the file name.

Usage:

```
EnvManager$get_data_fn(fn, dfn)
```

Arguments:

`fn` The file name.

`dfn` The name of the default file in the external data folder of the package.

Returns: The name of the file if it exists, or the full path to the default file.

Method `remove_files()`**:** Removes all files in the given directory.

Usage:

```
EnvManager$remove_files(dn)
```

Arguments:

`dn` The directory name.

Method `td_env()`**:** Removes the ed folder created by the setup_env method. Also sets the R option, "ed" to NULL.

Usage:

```
EnvManager$td_env(rf = F)
```

Arguments:

`rf` If the environment folder should be removed.

Method `cp_env()`**:** Copies the ed folder created by the setup_env method to inst/extdata.

Usage:

```
EnvManager$cp_env()
```

Method `setup_env()`**:** Copies the given files from test folder to the environment folder.

Usage:

```
EnvManager$setup_env(fns = c(), cf = NULL)
```

Arguments:

`fns` The list of test files to copy

`cf` A custom environment folder. It is a path relative to the current directory. If not specified, then the tempdir function is used to generate the environment folder.

Returns: The list of folders that can be used during testing.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EnvManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Model` *Represents n-gram models*

Description

The Model class represents n-gram models. An instance of the class is a single n-gram model. The attributes of this class are used to store n-gram model information. The class provides methods for loading and saving the model.

Details

The attributes of this class are used to store n-gram model information such as model name, model description, model file name, n-gram size, transition probabilities data, default probability for words, data cleaning and tokenization options, word list, model path, data directory path and performance stats. The model is saved to a single file as a R object.

A model file contains all the information required by the model. The model object is used as input by classes that perform operations on the model such as evaluation of model performance, text predictions and comparison of model performance.

Super class

`wordpredictor::Base` -> `Model`

Public fields

`pstats` The performance stats for the model.

`name` The model name.

`desc` The model description.

Methods

Public methods:

- `Model$new()`
- `Model$load_model()`
- `Model$get_config()`
- `Model$get_size()`
- `Model$clone()`

Method `new()`**:** It initializes the current object. It is used to set the maximum n-gram number, sample size, input file name, data cleaner options, tokenization options, combined transition probabilities file name and verbose.

Usage:

```
Model$new(
  name = NULL,
  desc = NULL,
  fn = NULL,
  df = NULL,
  n = 4,
  ssize = 0.3,
  dir = ".",
  dc_opts = list(),
  tg_opts = list(),
  ve = 0
)
```

Arguments:

`name` The model name.

`desc` The model description.

`fn` The model file name.

`df` The name of the file used to generate the model.

`n` The maximum n-gram number supported by the model.

`ssize` The sample size as a proportion of the input file.

`dir` The directory containing the model files.

`dc_opts` The data cleaner options.

`tg_opts` The token generator options.

`ve` The level of detail in the information messages.

Method `load_model()`**:** It loads the model using the given information

Usage:

```
Model$load_model()
```

Method `get_config()`**:** It returns the given configuration data

Usage:

```
Model$get_config(cn)
```

Arguments:

`cn` The name of the required configuration.

Returns: The configuration value.

Method `get_size()`**:** It returns the size of the current object. The object size is calculated as the sum of sizes of the object attributes.

Usage:

```
Model$get_size()
```

Returns: The size of the object in bytes.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ModelEvaluator` *Evaluates performance of n-gram models*

Description

It provides methods for performing extrinsic and intrinsic evaluation of a n-gram model. It also provides a method for comparing performance of multiple n-gram models.

Intrinsic evaluation is based on calculation of Perplexity. Extrinsic evaluation involves determining the percentage of correct next word predictions.

Details

Before performing the intrinsic and extrinsic model evaluation, a validation file must be first generated. This can be done using the DataSampler class.

Each line in the validation file is evaluated. For intrinsic evaluation Perplexity for the line is calculated. An overall summary of the Perplexity calculations is returned. It includes the min, max and mean Perplexity.

For extrinsic evaluation, next word prediction is performed on each line. If the actual next word is one of the three predicted next words, then the prediction is considered to be accurate. The extrinsic evaluation returns the percentage of correct and incorrect predictions.

Super class

`wordpredictor::Base` -> `ModelEvaluator`

Methods

Public methods:

- `ModelEvaluator$new()`
- `ModelEvaluator$compare_performance()`
- `ModelEvaluator$plot_stats()`
- `ModelEvaluator$evaluate_performance()`
- `ModelEvaluator$intrinsic_evaluation()`
- `ModelEvaluator$extrinsic_evaluation()`
- `ModelEvaluator$clone()`

Method `new()`**:** It initializes the current object. It is used to set the model file name and verbose options.

Usage:

```
ModelEvaluator$new(mf = NULL, ve = 0)
```

Arguments:

`mf` The model file name.

`ve` The level of detail in the information messages.

Method `compare_performance()`**:** It compares the performance of the models in the given folder.

The performance of the model is compared for the 4 metric which are time taken, memory used, Perplexity and accuracy. The performance comparison is displayed on plots.

4 plots are displayed. One for each performance metric. A fifth plot shows the variation of Perplexity with accuracy. All 5 plots are plotted on one page.

Usage:

```
ModelEvaluator$compare_performance(opts)
```

Arguments:

`opts` The options for comparing model performance.

- **save_to**. The graphics device to save the plot to. NULL implies plot is printed.
- **dir**. The directory containing the model file, plot and stats.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be
# used.
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code
# ModelEvaluator class object is created
me <- ModelEvaluator$new(ve = ve)
# The performance evaluation is performed
me$compare_performance(opts = list(
    "save_to" = NULL,
    "dir" = ed
))

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `plot_stats()`**:** It plots the given stats on 5 plots. The plots are displayed on a single page.

The 4 performance metrics which are time taken, memory, Perplexity and accuracy are plotted against the model name. Another plot compares Perplexity with accuracy for each model.

Usage:

```
ModelEvaluator$plot_stats(data)
```

Arguments:

`data` The data to plot

Returns: The ggplot object is returned.

Method `evaluate_performance()`**:** It performs intrinsic and extrinsic evaluation for the given model and validation text file. The given number of lines in the validation file are used in the evaluation

It performs two types of evaluations. One is intrinsic evaluation, based on Perplexity, the other is extrinsic evaluation based on accuracy.

It returns the results of evaluation. 4 evaluation metrics are returned. Perplexity, accuracy, memory and time taken. Memory is the size of the model object. Time taken is the time needed for performing both evaluations.

The results of the model evaluation are saved within the model object and also returned.

Usage:

```
ModelEvaluator$evaluate_performance(lc, fn)
```

Arguments:

`lc` The number of lines of text in the validation file to be used for the evaluation.

`fn` The name of the validation file. If it does not exist, then the default file validation-clean.txt is checked in the models folder

Returns: The performance stats are returned.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The performance evaluation is performed
stats <- me$evaluate_performance(lc = 20, fn = vfn)
# The evaluation stats are printed
```

```
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `intrinsic_evaluation()`**:** Evaluates the model using intrinsic evaluation based on Perplexity. The given number of sentences are taken from the validation file. For each sentence, the Perplexity is calculated.

Usage:

```
ModelEvaluator$intrinsic_evaluation(lc, fn)
```

Arguments:

`lc` The number of lines of text in the validation file to be used for the evaluation.

`fn` The name of the validation file. If it does not exist, then the default file validation-clean.txt is checked in the models folder

Returns: The min, max and mean Perplexity score.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The intrinsic evaluation is performed
stats <- me$intrinsic_evaluation(lc = 20, fn = vfn)
# The evaluation stats are printed
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `extrinsic_evaluation()`**:** Evaluates the model using extrinsic evaluation based on Accuracy. The given number of sentences are taken from the validation file.
For each sentence, the model is used to predict the next word. The accuracy stats are returned. A prediction is considered to be correct if one of the predicted words matches the actual word.

Usage:

```
ModelEvaluator$extrinsic_evaluation(lc, fn)
```

Arguments:

`lc` The number of lines of text in the validation file to be used for the evaluation.

`fn` The name of the validation file.

Returns: The number of correct and incorrect predictions.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The intrinsic evaluation is performed
stats <- me$extrinsic_evaluation(lc = 100, fn = vfn)
# The evaluation stats are printed
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelEvaluator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ModelEvaluator$compare_performance`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be
# used.
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code
# ModelEvaluator class object is created
me <- ModelEvaluator$new(ve = ve)
# The performance evaluation is performed
me$compare_performance(opts = list(
    "save_to" = NULL,
    "dir" = ed
))

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `ModelEvaluator$evaluate_performance`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
```

```
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The performance evaluation is performed
stats <- me$evaluate_performance(lc = 20, fn = vfn)
# The evaluation stats are printed
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `ModelEvaluator$intrinsic_evaluation`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The intrinsic evaluation is performed
stats <- me$intrinsic_evaluation(lc = 20, fn = vfn)
# The evaluation stats are printed
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `ModelEvaluator$extrinsic_evaluation`
## ------------------------------------------------
```

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS", "validate-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# The validation file name
vfn <- paste0(ed, "/validate-clean.txt")

# ModelEvaluator class object is created
me <- ModelEvaluator$new(mf = mfn, ve = ve)
# The intrinsic evaluation is performed
stats <- me$extrinsic_evaluation(lc = 100, fn = vfn)
# The evaluation stats are printed
print(stats)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

`ModelGenerator` *Generates n-gram models from a text file*

Description

It provides a method for generating n-gram models. The n-gram models may be customized by specifying data cleaning and tokenization options.

Details

It provides a method that generates a n-gram model. The n-gram model may be customized by specifying the data cleaning and tokenization options.

The data cleaning options include removal of punctuation, stop words, extra space, non-dictionary words and bad words. The tokenization options include n-gram number and word stemming.

Super class

`wordpredictor::Base` -> `ModelGenerator`

Methods

Public methods:

- ModelGenerator$new()
- ModelGenerator$generate_model()
- ModelGenerator$clone()

Method `new()`**:** It initializes the current object. It is used to set the maximum n-gram number, sample size, input file name, data cleaner options, tokenization options and verbose option.

Usage:

```
ModelGenerator$new(
  name = NULL,
  desc = NULL,
  fn = NULL,
  df = NULL,
  n = 4,
  ssize = 0.3,
  dir = ".",
  dc_opts = list(),
  tg_opts = list(),
  ve = 0
)
```

Arguments:

`name` The model name.

`desc` The model description.

`fn` The model file name.

`df` The path of the input text file. It should be the short file name and should be present in the data directory.

`n` The n-gram size of the model.

`ssize` The sample size as a proportion of the input file.

`dir` The directory containing the input and output files.

`dc_opts` The data cleaner options.

`tg_opts` The token generator options.

`ve` The level of detail in the information messages.

Method `generate_model()`**:** It generates the model using the parameters passed to the object's constructor. It generates a n-gram model file and saves it to the model directory.

Usage:

```
ModelGenerator$generate_model()
```

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
```

```
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# ModelGenerator class object is created
mg <- ModelGenerator$new(
    name = "default-model",
    desc = "1 MB size and default options",
    fn = "def-model.RDS",
    df = "input.txt",
    n = 4,
    ssize = 0.99,
    dir = ed,
    dc_opts = list(),
    tg_opts = list(),
    ve = ve
)
# The n-gram model is generated
mg$generate_model()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelGenerator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ModelGenerator$generate_model`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
```

```
# package
rf <- c("input.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# ModelGenerator class object is created
mg <- ModelGenerator$new(
    name = "default-model",
    desc = "1 MB size and default options",
    fn = "def-model.RDS",
    df = "input.txt",
    n = 4,
    ssize = 0.99,
    dir = ed,
    dc_opts = list(),
    tg_opts = list(),
    ve = ve
)
# The n-gram model is generated
mg$generate_model()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

`ModelPredictor` *Allows predicting text, calculating word probabilities and Perplexity*

Description

It provides a method for predicting the new word given a set of previous words. It also provides a method for calculating the Perplexity score for a set of words. Furthermore it provides a method for calculating the probability of a given word and set of previous words.

Super class

`wordpredictor::Base` -> `ModelPredictor`

Methods

Public methods:

- `ModelPredictor$new()`
- `ModelPredictor$get_model()`
- `ModelPredictor$calc_perplexity()`
- `ModelPredictor$predict_word()`

- ModelPredictor$get_word_prob()
- ModelPredictor$clone()

Method `new()`**:** It initializes the current object. It is used to set the model file name and verbose options.

Usage:

```
ModelPredictor$new(mf, ve = 0)
```

Arguments:

`mf` The model file name.

`ve` The level of detail in the information messages.

Method `get_model()`**:** Returns the Model class object.

Usage:

```
ModelPredictor$get_model()
```

Returns: The Model class object is returned.

Method `calc_perplexity()`**:** The Perplexity for the given sentence is calculated. For each word, the probability of the word given the previous words is calculated. The probabilities are multiplied and then inverted. The nth root of the result is the perplexity, where n is the number of words in the sentence. If the stem_words tokenization option was specified when creating the given model file, then the previous words are converted to their stems.

Usage:

```
ModelPredictor$calc_perplexity(words)
```

Arguments:

`words` The list of words.

Returns: The perplexity of the given list of words.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
```

```
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The sentence whoose Perplexity is to be calculated
l <- "last year at this time i was preparing for a trip to rome"
# The line is split in to words
w <- strsplit(l, " ")[[1]]
# The Perplexity of the sentence is calculated
p <- mp$calc_perplexity(w)
# The sentence Perplexity is printed
print(p)
# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `predict_word()`**:** Predicts the next word given a list of previous words. It checks the last n previous words in the transition probabilities data, where n is equal to 1 - n-gram size of model. If there is a match, the top 3 next words with highest probabilities are returned. If there is no match, then the last n-1 previous words are checked. This process is continued until the last word is checked. If there is no match, then empty result is returned. The given words may optionally be stemmed.

Usage:

```
ModelPredictor$predict_word(words, count = 3, dc = NULL)
```

Arguments:

`words` A character vector of previous words or a single vector containing the previous word text.

`count` The number of results to return.

`dc` A DataCleaner object. If it is given, then the given words

Returns: The top 3 predicted words along with their probabilities.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, "rp" = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
```

```
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The next word is predicted
nws <- mp$predict_word("today is", count = 10)
# The predicted next words are printed
print(nws)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `get_word_prob()`**:** Calculates the probability of the given word given the previous words. The last n words are converted to numeric hash using digest2int function. All other words are ignored. n is equal to 1 - size of the n-gram model. The hash is looked up in a data frame of transition probabilities. The last word is converted to a number by checking its position in a list of unique words. If the hash and the word position were found, then the probability of the previous word and hash is returned. If it was not found, then the hash of the n-1 previous words is taken and the processed is repeated. If the data was not found in the data frame, then the word probability is returned. This is known as back-off. If the word probability could not be found then the default probability is returned. The default probability is calculated as 1/(N+V), Where N = number of words in corpus and V is the number of dictionary words.

Usage:

```
ModelPredictor$get_word_prob(word, pw)
```

Arguments:

`word` The word whose probability is to be calculated.

`pw` The previous words.

Returns: The probability of the word given the previous words.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, "rp" = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The probability that the next word is "you" given the prev words
```

```
# "how" and "are"
prob <- mp$get_word_prob(word = "you", pw = c("how", "are"))
# The probability is printed
print(prob)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelPredictor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ModelPredictor$calc_perplexity`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The sentence whoose Perplexity is to be calculated
l <- "last year at this time i was preparing for a trip to rome"
# The line is split in to words
w <- strsplit(l, " ")[[1]]
# The Perplexity of the sentence is calculated
p <- mp$calc_perplexity(w)
# The sentence Perplexity is printed
print(p)
# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

```
## ------------------------------------------------
## Method `ModelPredictor$predict_word`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, "rp" = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The next word is predicted
nws <- mp$predict_word("today is", count = 10)
# The predicted next words are printed
print(nws)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()

## ------------------------------------------------
## Method `ModelPredictor$get_word_prob`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("def-model.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, "rp" = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The model file name
```

```
mfn <- paste0(ed, "/def-model.RDS")
# ModelPredictor class object is created
mp <- ModelPredictor$new(mf = mfn, ve = ve)
# The probability that the next word is "you" given the prev words
# "how" and "are"
prob <- mp$get_word_prob(word = "you", pw = c("how", "are"))
# The probability is printed
print(prob)

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

`TokenGenerator` *Generates n-grams from text files*

Description

It generates n-gram tokens along with their frequencies. The data may be saved to a file in plain text format or as a R object.

Super class

`wordpredictor::Base` -> `TokenGenerator`

Methods

Public methods:

- `TokenGenerator$new()`
- `TokenGenerator$generate_tokens()`
- `TokenGenerator$clone()`

Method `new()`**:** It initializes the current obj. It is used to set the file name, tokenization options and verbose option.

Usage:

```
TokenGenerator$new(fn = NULL, opts = list(), ve = 0)
```

Arguments:

`fn` The path to the input file.

`opts` The options for generating the n-gram tokens.

- **n**. The n-gram size.
- **save_ngrams**. If the n-gram data should be saved.
- **min_freq**. All n-grams with frequency less than min_freq are ignored.
- **line_count**. The number of lines to process at a time.
- **stem_words**. If words should be transformed to their stems.
- **dir**. The dir where the output file should be saved.

- **format**. The format for the output. There are two options.
  - **plain**. The data is stored in plain text.
  - **obj**. The data is stored as a R obj.

ve The level of detail in the information messages.

Method `generate_tokens()`**:** It generates n-gram tokens and their frequencies from the given file name. The tokens may be saved to a text file as plain text or a R object.

Usage:

```
TokenGenerator$generate_tokens()
```

Returns: The data frame containing n-gram tokens along with their frequencies.

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram size
n <- 4
# The test file name
tfn <- paste0(ed, "/test-clean.txt")
# The n-gram number is set
tg_opts <- list("n" = n, "save_ngrams" = TRUE, "dir" = ed)
# The TokenGenerator object is created
tg <- TokenGenerator$new(tfn, tg_opts, ve = ve)
# The n-gram tokens are generated
tg$generate_tokens()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TokenGenerator$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `TokenGenerator$generate_tokens`
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("test-clean.txt")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The n-gram size
n <- 4
# The test file name
tfn <- paste0(ed, "/test-clean.txt")
# The n-gram number is set
tg_opts <- list("n" = n, "save_ngrams" = TRUE, "dir" = ed)
# The TokenGenerator object is created
tg <- TokenGenerator$new(tfn, tg_opts, ve = ve)
# The n-gram tokens are generated
tg$generate_tokens()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

`TPGenerator` *Generates transition probabilities for n-grams*

Description

It provides a method for generating transition probabilities for the given n-gram size. It also provides a method for generating the combined transition probabilities data for n-gram sizes from 1 to the given size. The combined transition probabilities data can be used to implement back-off.

Details

It provides a method for generating n-gram transition probabilities. It reads n-gram frequencies from an input text file that is generated by the TokenGenerator class.

It parses each n-gram into a prefix, a next word, the next word frequency and the next word probability. Maximum Likelihood count is used to generate the next word probabilities.

Each n-gram prefix is converted to a numeric hash using the digest2int function. The next word is replaced with the position of the next word in the list of all words. The transition probabilities data is stored as a dataframe in a file.

Another method is provided that combines the transition probabilities for n-grams of size 1 to the given size. The combined transition probabilities can be saved to a file as a data frame. This file may be regarded as a completed self contained n-gram model. By combining the transition probabilities of n-grams, back-off may be used to evaluate word probabilities or predict the next word.

Super class

`wordpredictor::Base` -> `TPGenerator`

Methods

Public methods:

- `TPGenerator$new()`
- `TPGenerator$generate_tp()`
- `TPGenerator$generate_tp_for_n()`
- `TPGenerator$clone()`

Method `new()`**:** It initializes the current obj. It is used to set the transition probabilities options and verbose option.

Usage:

```
TPGenerator$new(opts = list(), ve = 0)
```

Arguments:

`opts` The options for generating the transition probabilities.

- **save_tp**. If the data should be saved.
- **n**. The n-gram size.
- **dir**. The directory containing the input and output files.
- **format**. The format for the output. There are two options.
  - **plain**. The data is stored in plain text.
  - **obj**. The data is stored as a R obj.

`ve` The level of detail in the information messages.

Method `generate_tp()`**:** It first generates the transition probabilities for each n-gram of size from 1 to the given size. The transition probabilities are then combined into a single data frame and saved to the output folder that is given as parameter to the current object.

By combining the transition probabilities for all n-gram sizes from 1 to n, back-off can be used to calculate next word probabilities or predict the next word.

Usage:

```
TPGenerator$generate_tp()
```

Examples:

```
# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n1.RDS", "n2.RDS", "n3.RDS", "n4.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The list of output files
fns <- c("words", "model-4", "tp2", "tp3", "tp4")

# The TPGenerator object is created
tp <- TPGenerator$new(opts = list(n = 4, dir = ed), ve = ve)
# The combined transition probabilities are generated
tp$generate_tp()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Method `generate_tp_for_n()`**:** It generates the transition probabilities table for the given n-gram size. It first reads n-gram token frequencies from an input text file.

It then generates a data frame whose columns are the n-gram prefix, next word and next word frequency. The data frame may be saved to a file as plain text or as a R obj. If n = 1, then the list of words is saved.

Usage:

```
TPGenerator$generate_tp_for_n(n)
```

Arguments:

`n` The n-gram size for which the tp data is generated.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TPGenerator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `TPGenerator$generate_tp`
```

```
## ------------------------------------------------

# Start of environment setup code
# The level of detail in the information messages
ve <- 0
# The name of the folder that will contain all the files. It will be
# created in the current directory. NULL implies tempdir will be used
fn <- NULL
# The required files. They are default files that are part of the
# package
rf <- c("n1.RDS", "n2.RDS", "n3.RDS", "n4.RDS")
# An object of class EnvManager is created
em <- EnvManager$new(ve = ve, rp = "./")
# The required files are downloaded
ed <- em$setup_env(rf, fn)
# End of environment setup code

# The list of output files
fns <- c("words", "model-4", "tp2", "tp3", "tp4")

# The TPGenerator object is created
tp <- TPGenerator$new(opts = list(n = 4, dir = ed), ve = ve)
# The combined transition probabilities are generated
tp$generate_tp()

# The test environment is removed. Comment the below line, so the
# files generated by the function can be viewed
em$td_env()
```

Index