Tokenizer

Regex

The tokenizer uses this regexes to find the tokens:

TokenLabel	Regex
PUNCTUATION"	`r'[!"#$%&\\\'()*+,\\-./:;≈❗,«»<=>?@\\[\\\\\\\]\\^_`|~،”“.×·´—–؟⁝{}…]'`
"EMOJI"	`"["+open(join(DIRNAME,"./emojies")).read().replace("\n","")+"]"`
"IP"	`r"\d\.\d\.\d\.\d"`
"NUMBER"	`r"(?:+(?:(?:.+)`
"PERSIAN_WORD"	`f"[{persianLetters}][{persianLetters}\u200c]*[{persianLetters}]|[{persianLetters}]"`
"ENGLISH_WORD"	`r"[A-Za-z]+(?:\'[a-z]+)?"`
"SENTENCE_DELIMITERS"	`r"[.?!؟]"`
"ENGLISH_HUMAN_NAME"	`r"(?:(?i)mr|mis|miss)\.[A-Za-z][a-z]+"`
"EMAIL"	`r"[a-zA-Z0-9.!#$%&'*+\/=?^_{|}~-]+ @ (?: [a-zA-Z0-9] [-a-zA-Z0-9%_\+~#=]* \. )+ [a-zA-Z] [a-zA-Z0-9]{0,6}"`
"LINK"	`r"(?:https?:\/\/)? (?:www\d?\.)? (?: [a-zA-Z0-9] [-a-zA-Z0-9%_\+~#=]* \. )+ [a-zA-Z] [a-zA-Z0-9]{0,6} (?: / [a-zA-Z0-9_%+=~\-#@]+ (?: \. [a-zA-Z0-9_%+=~\-@]+ )* )* \/? (?: \? (?: [a-zA-Z] [a-zA-Z0-9]* = [a-zA-Z0-9+\.\-%]* & )* (?: [a-zA-Z] [a-zA-Z0-9]* = [a-zA-Z0-9+\.\-%]* ) )?"`
"GREEL_LETTERS"	`"[ΑαΒβΓγŋΔδΕεΖζΗηΘθΙιΚκΛλΜμΝνΞξΟοΠπΡρΣσΤτΥυΦφΧχΨψΩω∑∈]"`
"DEGREE"	`r"(\d+°C)|(\d+°F)"`
"LEFT_TO_RIGHT_MARK "	`"\u200e"`
"RIGHT_TO_LEFT_MARK "	`"\u200f"`
"NEW_LINE"	`r"\n"`
"SPACE"	`" "`
"TAB"	`r"\t"`

> emojies are stored in a file named `emojies` which is available in ./pltk/Tokenizer/emojies

> persianLetters is a variable which contains all of the persian letters

> DOT OPERATOR' (U+22C5) ==> ⋅ ,PRIME (U+2032) ==> ′ ,MINUS SIGN (U+2212) ==> − ,EN DASH (U+2013) ==> – ,ARABIC SEMICOLON' (U+061B) ,arabic question mark (U+061F) ,FUNCTION APPLICATION (U+2061) are also supported as `PUNCTUATION`

Normalizer

the normalizer conversions are described bellow:

The characters in column A converts to characters in column B

A	B
١	۱
٢	۲
٣	۳
٤	۴
٥	۵
٦	۶
٧	۷
٨	۸
٩	۹
٪	%
٫	.
٬	،
٭	*
ڨڤ	ق
ﺎ ٱ ٲ ٳ ٵ ﺃﺇا	ا
ﺁ	آ
ۈ ۊ ۋ ﯗ ۆ ﯙ ۉ ۏ ﯛ ۅ ۇ ﺅ ۄ وﻭ	و
ٺٽ	ث
ﺑﺒٻ	ب
ٹٺټٿﺘﺗ	ت
ﭗﭙ	پ
ځ	ح
ﺥﺦﺧﺨڅ	خ
ڇڿ ﭽ	چ
ﺟﺠڃڄ	ج
ﺣﺤ	ح
ډڊڋڍﺪ	د
ڈڌڎڏڐ	ذ
ڗڙﮋ	ژ
ڒڔڕږﺮ	ر
ﺰﺯړ	ز
ښڛﺳﺴس	س
ۺڜﺷﺸش	ش
ڝﺻﺼص	ص
ۻ	ض
ﻅ	ط
ڠۼﻐﻏ	غ
ﻌﻋ	ع
ڢڣڤڥڦﻓﻔف	ف
ڧ	ق
ﻙڪګڬڭڮﻛﻜک	ک
ڰڱڲڳڴﮔﮕگ	گ
ڵڶڷڸﻟﻠل	ل
ڹںڻڼڽﻧﻨ	ن
ﻁ	ط
ۀ ۃ ﮭﮬﮫﮪ ۂ ﻫ ﻪ ﻬ ھ	ه
ۂ	ع
ﻣﻤ	م
﷽﷼ﷺﷻ	_
ىﯨﯩﯼﯽﯾﯿئ ۍ ێ ې ۑ ے ۓ ﯤ ﯥ ﯦ ﯧ ﻱ ﻲ ﻳ ﻴ ي	ی
—ـ	_
،	,
"\u2063" (INVISIBLE SEPARATOR)	,
"\u2067" (RIGHT-TO-LEFT ISOLATE)	"\u200f" (RIGHT_TO_LEFT_MARK)
'\u200d'	'\u200c'
'\ufeff'	'\u200c'
'\u00a0'	'\u200c'

Coding

The tokenizer function is available in `pltk.Tokenizer.wordTokenizer`.

Arguments:

Arguments	Description
text	`string` a text to tokenize
tokensMap	`dict` *optional* A dictionary in which `Tokens Labels` point to `compiled regex patterns`(e.g. `re.compile` output) **default token map is described above**
Normalizer	`function`*optional* a function used for normalizing the text prior to tokenizing **default normalizer is described above**
verbose	`bool`*optional* shows a progress bar if `True`

Returns:
Returns two lists of `Token` class as detected tokens and characters which does not fit in any of the known regexes.

Testing Tokenizer on Telegram Data

The data extracted from telegram channels contains 21217 lines and 5258018 characters(this numbers are reported by wc command)

the `runme.py` program tokenize the the these data and counts the frequency of each token.All tokens are stored in `tokensCount.xlsx` file and the output is:

```
$ python3 runme.py
wordTokenizer: 100%|████████████████████| 21218/21218 [00:35<00:00, 597.51it/s]

wordCounter: 100%|████████████████| 1976270/1976270 [00:03<00:00, 622662.52it/s]
tokenizerFinished...
                                              type text  termFrequency
<TOKEN type:SPACE pos:(4, 5)>,[ ]                SPACE              851769
<TOKEN type:PUNCTUATION pos:(191, 192)>,[!]  PUNCTUATION    !           2569
<TOKEN type:PUNCTUATION pos:(35, 36)>,["]    PUNCTUATION    "           2200
<TOKEN type:PUNCTUATION pos:(3, 4)>,[#]      PUNCTUATION    #           3509
<TOKEN type:PUNCTUATION pos:(238, 239)>,[%]  PUNCTUATION    %             96
...                                            ...  ...            ...
<TOKEN type:EMOJI pos:(0, 1)>,[🏘]                EMOJI    🏘              1
<TOKEN type:EMOJI pos:(943, 944)>,[🌪]            EMOJI    🌪              1
<TOKEN type:EMOJI pos:(85, 86)>,[🎖]              EMOJI    🎖              3
<TOKEN type:EMOJI pos:(93, 94)>,[🎖]              EMOJI    🎖              3
<TOKEN type:EMOJI pos:(103, 104)>,[🎖]            EMOJI    🎖              3

[33763 rows x 3 columns]
```

As you can see the output contains 33763 unique tokens also the labels and term frequencies are specified in `tokensCount.xlsx`.

The program used 35 seconds for the tokenizing and 3 seconds for counting the tokens frequency.

calculating TF-IDF

Codes

Tokenizer

the tokenizer is descriped in last project

TF(documents,outputFormat='sparseMatrix')

- **Parameters:** documents: string, list an string contains a folder name of corpus or list of documents outputFormat: string 'sparseMatrix' or 'pandas_DataFrame'
- **Returns:**
  an sparse matrix or `pandas.DataFrame` object of term frequency calculated.

In order to calculate the TF of the documents, several steps are taken as descriped below: - Fistly, each file will be opened and read using `open` module. - Secondly, the data extracted from files are passed to `tokenizer.wordTokenizer` which is implemented in the previous assignment.this step will convert the text into the list of tokens. - In the third step,the list of tokens are passed to `tokenizer.wordCounter` in order to count the frequency of each token in the current document. This finction is also implemented in the previous assignment. The output will be a `dict` object which maps tokens into their frequency. - As the next step, The last calculated dictionary is stored as the value of another dictionary. This nested dictionary structures is forming a sparse matrix which saves more space than regular matrix. - Finally, the function will return the output matrix as `pandas.DataFrame` if needed.

DF(documents,outputFormat='sparseMatrix')

- **Parameters:** documents: string, list an string contains a folder name of corpus or list of documents outputFormat: string 'sparseMatrix' or 'pandas_DataFrame'
- **Returns:**
  map of tokens to their DocumentFrequency of `pandas.DataFrame`

In order to calculate the DF of the documents, several steps are taken as descriped below: - Fistly, each file will be opened and read using `open` module. - Secondly, the data extracted from files are passed to `tokenizer.wordTokenizer` which is implemented in the previous assignment.this step will convert the text into the list of tokens. - In the third step, the frequency of each token will increase by one value if that token is in the current document. - Finally, the function will return the output matrix as `pandas.DataFrame` if needed.

TF_IDF(TF_mat,DF_mat,outputFormat='sparseMatrix')

- Parameters:

```
TF_mat:
the output of TF function
DF_mat:
the output of DF or DF_fromTF function
outputFormat: string
'sparseMatrix' or 'pandas_DataFrame'
```

- Returns:
  an sparse matrix or `pandas.DataFrame` object of TF-IDF calculated.

Having TF and DF will makes calculation of the TF-IDF pretty easy. simply each row of the TF matrix is divided by the corresponding term in the DF matrix.

calculate DF matrix using TF matrix

ther is another faster way to calculate DF matrix and that is using TF matrix.To do so,number of non-zero values will be count in each line of the TF matrix. This operation is incredibly fast using `pandas.DataFrame`: - First: divide the TF matrix by itself - Second: add numbers in each line and save the answer as the DocumentFrequency of terms.

Test

There is two test corpus available: - CorpusSmall: contains two small files types by my self just to test the outputs. - CorpusBig: contains 150 file which each files has 16 lines.

Output

there is brief information in standard output containing execution time and small view of the matrixes.

Execution Time:

```
The TF matrix is calculated in 4.823282718658447

The DF matrix is calculated in 0.007876873016357422

The TF-IDF matrix is calculated in 0.15381646156311035
```

obviously calculating the TF requires the hard drive file access, which makes it slowly.

Saved Files

There is three saved files `tf.xlsx`, `df.xlsx`, `tf_idf.xlsx`.(their names are clear enough to decribe their contents)

tf.xlsx

5045 rows x 150 columns rows contains tokens. columns contains documents.

DF.xlsx

5046 tokens are found and sorted.

tf_idf.xlsx

5045 rows x 150 columns rows contains tokens. columns contains documents.