Linguistic Analyzer Documentation

Release 2.0

Paul Brown, Tyler Blanton

Mar 20, 2018

Contents:

CHAPTER 1

Keyword module

class Keyword.**Keyword**(*nWord=”*, *nType=0*, *nSal=0*, *nFreq=0*, *nKeyscore=0*)
Bases: `object`

summary: Stores a specific keyword and it’s associated information. The constructor accepts the word, type, salience, frequency and keyscore.

classmethod issimilar(*passedWord*)
summary: determines if the passed keyword is similar to (or exactly the same as) the main word in the class

Parameters **passedWord** (*str*) – word

Returns boolean value of True or False

Return type bool

similarwordfrequency()

Returns the frequency of a similar word in a document

Return type int

wordfrequency()

Returns the frequency value of a word

Return type int

CHAPTER 2

KeywordList module

class KeywordList.**KeywordList**

Bases: `object`

Summary: A list that contains keywords. The list also contains unique keyword value, keyword score, yules k score, yules score and a document score.

calculateavgscores()

Summary: calculates a document's average score values.

Returns void

existsinlist(*keyword_name*)

Summary: searches through the list of keywords and sees if any keywords shares the same Keyword.word.

Parameters **keyword_name** (*str*) – The keyword

Returns returns true if a keyword with keyword_name as Keyword.word exists in the list. False otherwise.

Return type bool

getavgkeywordscore()

Summary: returns document's average keyword score.

Returns average keyword score

Return type int

getdocumentscore()

Summary: Returns document's score.

Returns document score

Return type int

getindexofword(*keyword_name*)

Summary: returns index of a Keyword in the list of Keywords

Parameters **key_name** (*str*) – keyword

Returns keyword index

Return type int

getkeywordscore()

Summary: returns document's keyword score.

Returns keyword score of document

Return type int

getyulesiscore()

Summary: returns document's Yule's i score.

Returns Yule's I score

Return type int

getyuleskscore()

Summary: returns document's Yule's k score.

Returns Yules K score

Rytpe int

insertkeyword(*keyword*)

Summary: inserts new Keyword into Keyword list

Parameters **keyword** (`Keyword`) – an instance of the class keyword

Returns void

CHAPTER 3

functionsv1 package

3.1 common_functions module

common_functions.**changefileextension**(*regfilename*)
Summary: Changes the file name string from *.pdf* to *.txt*.

Parameters **regfilename** (*str*) – name of regulatory file

Returns string with .pdf file extension

Return type str

common_functions.**cleantext**(*text_list*)
Summary: Removes special characters from text.

Parameters **text_list** (*List[str]*) – a text string

Returns text_list with no special chars

Return type List[str]

common_functions.**createkeywordfromgoogleapientity**(*entity*, *file_text*)
Summary: Creates a Keyword from a single entity that is returned by the google NLP API.

- The 'name' or Keyword, Type of keyword, Salience score, and Frequency of the keyword is inputted in the *Keyword* object. A default value of *0* is used for the keyword score. Keyword score is determined further on in the app.
- The frequency of a keyword is found by using the *getwordfrequency()* function.

Parameters

- **entity** (*dict*) – Google API response entity object
- **file_text** (*List[str]*) – entire text of file

Returns Populated Keyword object

Return type *Keyword*

`common_functions.`**`extractkeywordfromtxt`**(*filename*)

Summary: This function will extract existing keyword information from *.txt file* and place it into the `KeywordList` object.

- This function will read in the first line of an existing *.txt* file and place the Yule's K, Yule's I, and average keyword score into a `KeywordList` object.
- For the second and remaining lines, the Keyword, Salience score, Frequency of keyword, and Keyword score are placed into a `Keyword` object, and then inserted in a `KeywordList` object via the `KeywordList.KeywordList.insertkeyword()` function.
- An **Exception** will raise if the extraction of keywords from a *.txt* file does not fully complete. This is an issue that occurs in the AWS Beanstalk environment. Running the app on a local machine should not throw the Exception, based on testing.

Parameters **`file`** (*`str`*) – location of .txt file

Returns keyword list in file

Return type *KeywordList*

Raises Exception

`common_functions.`**`extractmicrosoftdocxtext`**(*file*, *testdownload_folder=None*)

Summary: Extracts text from any *.docx* document and returns it.

Parameters

- **`file`** (*`fileStorage`*) – the file to save
- **`testdownload_folder`** (*`str`*) – Specific download folder is necessary

Returns file's text

Return type List[str]

`common_functions.`**`extractpdftext`**(*file*, *testdownload_folder=None*, *RegDoc=False*)

Summary: Extracts text from PDF document referenced in the given file argument using the `PDFMiner` python package.

The first part of the code assigns a "chunk" size via the value *NUM_SEND_CHARS* set in /applicationconfig.json. *chunk_size* is designated to break up a long string of text into a list of strings, if needed. The default setting for this allows for a single string of text.

Before starting the PDF text extraction, logging is disabled due to the amount of statements PDFMiner produces in the log.

Parameters

- **`file`** (*`fileStorage`*) – the PDF file to extract text from
- **`testdownload_folder`** (*`str`*) – specific download folder if necessary
- **`RegDoc`** (*`bool`*) – flag specifying whether this is a user doc or a regulatory doc

Returns file's text

Return type List[str]

Raises FileNotFoundError

`common_functions.`**`generatebubblecsv`**(*kw_list*, *reg_kw_list*)

Summary: Creates a new *csv* file with all the keywords. The *csv* file is used to generate the Bubble Chart.

Parameters

- **kw_list** (KeywordList) – list of doc keywords
- **reg_kw_list** (KeywordList) – list of reg doc keywords

Returns void

Raises Exception

`common_functions.`**`geterrorpage`**(*errtext='Unknown Error'*)

Summary: Populates error message with proper response and returns html

Parameters **errtext** (*str*) – text of error

Returns html page with error displayed

Return type str

`common_functions.`**`getregulatorydoctext`**(*filename*)

Summary: Looks in the '/RegulatoryDocuments' folder for the file with the given *filename* and return's its text as a list of strings. The `extractpdftext()` function is utilized.

Parameters **filename** (*str*) – name of regulatory file without file ending on it

Returns file text

Return type List[str]

Raises FileNotFoundError

`common_functions.`**`getscorepage`**(*kw_list*, *reg_kw_list*, *userdocwordcount*, *filename*, *regfilename*)

Summary: Returns 'views/score_response.html' page that is populated with proper calculated Keyword, Comparison, and Yule's scores.

Parameters

- **kw_list** (KeywordList) – list of user document's Keyword objects
- **reg_kw_list** (KeywordList) – list of regulatory document's Keywords
- **userdocwordcount** (*int*) – word count of user document
- **filename** (*str*) – user document's file name
- **regfilename** (*str*) – regulatory document's file name

Returns html page with scores displayed

Return type str

`common_functions.`**`getwordfrequency`**(*word*, *file_text*)

Summary: Determines frequency of the given word in the file's text

Parameters

- **word** (*str*) – Word to find frequency of
- **file_text** (*List[str]*) – list of string containing entire text of file

Returns frequency of word parameter in text

Return type int

`common_functions.`**`homeCount`**()

Initializes variables for logging session

Returns void

`common_functions.`**`interpretexistingfile`**(*regfilename*)

Summary: Function that handles a newly uploaded regulatory doc or an existing regulatory document. The following occurs:

1. Checks *regfilename* to determine if it's a .pdf. If it's a .pdf, then that means that this is the first time a regulatory document has been published to the app. Analysis of the document needs to occur.
   - Extracts text from a pdf file with `getregulatorydoctext()`.
   - Identifies keywords with `analyze_functions.identifykeywords()`.
   - Calculates various scores for each keyword with `analyze_functions.calculatescores()` and `KeywordList.KeywordList.calculateavgscores()`.
   - The file extension is changed from *.pdf* to *.txt*. This is done so that the existing filename with extension can be used when exporting information to *.txt* file.
   - Exports keyword information to a *.txt* file via `outputkeywordtotext()`.
   - 'views/index.html' is edited to include the new regulatory document file path for future selection on the app home page. The file path points to the *.txt* version of the document so analysis does not occur again.
2. If the *regfilename* is not a *pdf* file, then it's a *txt* file. Analysis of the file does not need to occur since it was previously done.
   - The keywords with associated information are extracted from the *txt* file via `extractkeywordfromtxt()`.

Parameters **`regfilename`** (*`str`*) – name of regulatory file

Returns Keyword list of regulatory document.

Return type *KeywordList*

`common_functions.`**`interpretfile`**(*file*, *localuploadfolder*)

Summary: Function that handles uploaded user document. The following occurs:

1. Extracts text from a pdf file with `extractpdftext()`.
2. Identifies keywords with `analyze_functions.identifykeywords()`.
3. Calculates various scores for each keyword with `analyze_functions.calculatescores()` and `KeywordList.KeywordList.calculateavgscores()`.
4. Exports keyword information to a *.txt* file via `outputkeywordtotext()`.
5. Determines total word count for a *file*. The value is stored in the variable *wordcount*

Parameters

- **`file`** (*`fileStorage`*) – file to be interpreted
- **`localuploadfolder`** (*`str`*) – Place to temporary store file so it can be read from

Returns list of file's Keywords, wordcount

Return type *KeywordList*, int

`common_functions.`**`kwhighestfrequencies`**(*keyword_list*, *numtopkws=10*)

Summary: Returns the top 10 most frequent Keywords in an uploaded file.

Parameters

- **`keyword_list`** (`KeywordList`) – List of Keyword objects

- **numtopkws** (*int*) – number of keywords to return.

Returns Keywords with highest frequencies

Return type List[*Keyword*]

`common_functions.`**`kwhighestkeyscores`**(*keyword_list*)

Summary: Returns ten Keywords with the highest Keyword scores

Parameters **keyword_list** (`KeywordList`) – list of Keyword objects

Returns list of top keyword scores

Return type List[*Keyword*]

`common_functions.`**`longstringtostringlist`**(*longstring*, *strsize*)

Summary: This functions splits a long string *longstring* into strings of size *strsize* and returns a list of those strings.

Parameters

- **longstring** (*str*) – text of file
- **strsize** (*int*) – requested length of each string in created list of strings

Returns return_list

Return type List[str]

`common_functions.`**`outputkeywordtotext`**(*keylist*, *download_folder='Documents/Keywords.txt'*)

Summary: This function will write Keywords and associated data from an analyzed document to a *.txt* file.

- The first line written to the file is the Yule's K, Yule's I and average keyscore of a *`KeywordList`* object.
- The second and following lines include the Keyword, Salience score, Frequency of keyword, and the Keyword score from the *`Keyword`* object.
- An **Exception** will raise if the output of Keywords to a *.txt* file does not fully complete. This is an issue that occurs in the AWS Beanstalk environment. Running the app on a local machine should not throw the Exception, based on testing.

Parameters

- **keylist** (`KeywordList`) – list of document keywords
- **download_folder** (*str*) – location to save the *.txt* file.

Returns void

Raises Exception

`common_functions.`**`plotkeywordfrequency`**(*keyword_list1*, *keyword_list2*, *doc1name='doc1'*, *doc2name='doc2'*)

Summary: Plots keyword score of most frequently used keywords. Pulls keywords from *keyword_list1* and compares against *keyword_list2*. The `matplotlib` python package is used to aide in plotting the data.

If there are no common keywords to plot, a blank plot will display with the title "No Common Keywords to Plot".

Parameters

- **keyword_list1** (`KeywordList`) – user document keywords
- **keyword_list2** (`KeywordList`) – regulatory document keywords
- **doc1name** (*str*) – name of user document

- **doc2name** (`str`) – name of regulatory document

Returns void

`common_functions.`**`plotkeywordsalience`**(*keyword_list1*, *keyword_list2*, *doc1name='doc1'*, *doc2name='doc2'*)

Summary: Plots salience scores of most frequently used keywords. Pulls keywords from *keyword_list1* and compares against *keyword_list2*. The `matplotlib` python package is used to aide in plotting the data.

If there are no common keywords to plot, a blank plot will display with the title "No Common Keywords to Plot".

Parameters

- **keyword_list1** (`KeywordList`) – user KeywordList
- **keyword_list2** (`KeywordList`) – regulatory KeywordList
- **doc1name** (`str`) – user document name
- **doc2name** (`str`) – regulatory document name

Returns void

`common_functions.`**`plotkeywordscores`**(*keyword_list1*, *keyword_list2*, *doc1name='doc1'*, *doc2name='doc2'*)

Summary: Plots keyword score of the most frequently used keywords. Pulls keywords from *keyword_list1* and compares against *keyword_list2*. The `matplotlib` python package is used to aide in plotting the data.

If there are no common keywords to plot, a blank plot will display with the title "No Common Keywords to Plot".

Parameters

- **keyword_list1** (`KeywordList`) – user KeywordList
- **keyword_list2** (`KeywordList`) – regulatory KeywordList
- **doc1name** (`str`) – user document name
- **doc2name** (`str`) – regulatory document name

Returns void

`common_functions.`**`printStringList`**(*textList*)

Summary: Helper function that prints a list of strings

Parameters **textList** (`List[str]`) – a text string

Returns void

`common_functions.`**`printanalytics`**(*filename*, *regfilename*, *keywordlist*, *regkeywordlist*, *calctime*)

Summary: Saves the data passed in the argument to the ever-increasing file '/downloads/Analytics.txt' that contains data analytics information. Analytic information includes:

- Date/Time
- Processing time
- user doc file name and number of keywords
- regulatory doc file name and number of keywords

Parameters

- **filename** (`str`) – name of user document file
- **regfilename** (`str`) – name of regulatory document file

- **keywordlist** (KeywordList) – user document Keyword list
- **regkeywordlist** (KeywordList) – regulatory document Keyword list.
- **calctime** (*int*) – processing time of app.

Returns void

common_functions.**savefile**(*file*, *download_folder=None*)

Summary: Save's given file to '/Downloads' folder.

Parameters

- **file** (*fileStorage*) – the file to save
- **download_folder** (*str*) – specific download folder if necessary

Returns void

common_functions.**splitintosize**(*file_text*)

Summary: This function splits a list of keywords of any length into a list of keywords each of length specified by NUM_SEND_CHARS in '/applicationconfig.json'

Parameters **file_text** (*list*) – list of document's words

Returns file_text

Return type List[str]

common_functions.**stringlisttolonglongstring**(*string_list*)

Summary: Helper function to turn a list of strings into one long long string.

Parameters **string_list** (*List[str]*) – a string of text

Returns file's text

Return type long str

common_functions.**writeToConfig**(*key*, *value*)

Summary: Writes *value* into the '/applicationconfig.json' file.

Parameters

- **key** (*str*) – variable in which *value* is being written to
- **value** – value

Returns none

3.2 analyze_functions module

analyze_functions.**calculatecomparisonscore**(*kw_list*, *reg_kw_list*)

Summary: Compares the calculated scores of the two documents and generates value based on that comparison.

1. The top 10 Keywords with the highest frequency is gathered from the user document.
2. The top 10% of the regulatory document Keywords are gathered.
3. For the top 10 Keywords in the user document, if they are in the top 10% of words in the regulatory document, a value of '1' is added to a variable called *tempscore*.
4. *tempscore* / top 10% of reg doc keywords = the new *tempscore*
5. The final score that is returned: 100 - [abs(average keyword score of user doc - average keyword score of reg doc)] * *tempscore*

Parameters

- **kw_list** (KeywordList) – list of Keywords
- **reg_kw_list** (KeywordList) – list of Keywords

Returns comparison score of two documents

Return type float

analyze_functions.**calculatekeywordscore**(*kw_list*, *kw*)

Summary: calculate a keyword score for a single keyword. The current algorithm utilized is: [(keyword salience * keyword frequency) / (total keywords)] * 1000. Since the salience and frequency of a particular keyword is important to the overall feel of a document, these values are used to calculate the score.

Parameters

- **kw_list** (KeywordList) – all Keywords of a document.
- **kw** (Keyword) – keyword

Returns keyword score

Return type float

analyze_functions.**calculatescores**(*kw_list*, *file_text*)

Summary: function that calls *calculatekeywordscore()* and *calculateyulesscore()* and inputs those values into *Keyword* and *KeywordList* respectively for a particular document.

Parameters

- **kw_list** (KeywordList) – list of Keywords
- **file_text** (*List[str]*) – Text of file

Returns void

analyze_functions.**calculateyulesscore**(*file_text*)

Summary: calculates Yule's K/I scores for a given document. These scores are used to determine the lexical richness of a given document.

This function starts by ensuring that *file_text* is converted into a long string vice a list of strings to ensure accurate calculation of the scores. Then, the string is split into tokens via *tokenize()*. The Yule's K/I algorithm is implemented based on the tokens provided. If there is a *'Division by Zero'* error, an exception will be raised and the default score value will be **'-1'**

Parameters **file_text** (*List[str]*) – plain text of document

Returns Yules score of text file [Yule's K, Yule's I]

Return type float

Raises ZeroDivisionError

analyze_functions.**declarelogger**()

Summary: Declares logger for the current session. Logging statements are re-directed to a local logging file. The logging level is set to DEBUG.

LOG_FILE_PATH = 'logging/Linguistic_Analyzer.log'

analyze_functions.**identifykeywords**(*file_text*)

Summary: Calls the Google NLP API to extract Keyword information from text. The 'analyze entities' from the API is utilized. The information retained from the API is 'entity' (keyword) and the 'salience' value of a particular keyword.

Information regarding the Google NLP API can be found at: https://cloud.google.com/natural-language/

For use on a local machine: add export API_KEY="your API key" in bash.profile or whichever file contains environmental variable setup.

For use in AWS: enter 'API_KEY' with key value in AWS configuration settings

file_text contains the text of a particular document in a list of strings. The original idea here was concern that a long string of text would crash the app due to memory constraints. However, if document text is broken up and sent to the API as such, the analysis would not encompass the document in its entirety. Instead, the scores provided would be focused on each 'chunk' of text. Therefore, analysis of an entire document would be inaccurate. The list of strings idea here has remained, but the 'chunk' size for *file_text* can be configured in /applicationconfig.json. Default settings allow for a single string text input of a document into the API.

For each entity identified by the API, `common_functions.createkeywordfromgoogleapientity()` is used to extract the information from the *entities* dictionary variable and places it into a `Keyword`. The returned Keyword is then placed into the `KeywordList` object via `KeywordList.KeywordList.insertkeyword()`.

Parameters **`file_text`** (*`List[str]`*) – text of document

Returns KeywordList object

Return type *KeywordList*

Raises Exception

`analyze_functions.`**`tokenize`**(*tokenStr*)

Summary: Splits up string into individual tokens.

Parameters **`tokenStr`** (*`str`*) – a string of words

Returns tokens

Return type list

analyze module

`analyze.`**`analyzeText`**(*fileText*)

Parameters **`fileText`** (*`str`*) – text of fileText

Returns file text

Return type str

`analyze.`**`checkSimilarity`**(*fileText*)

Parameters **`fileText`** (*`str`*) – text of file

Returns pass or fail

Return type bool

`analyze.`**`createObjects`**(*fileText*)

Parameters **`fileText`** (*`str`*) – text of file

Returns pass or fail

Return type bool

`analyze.`**`scrapeText`**(*fileText*)

Parameters **`fileText`** (*`str`*) – text of file

Returns pass or fail

Return type bool

CHAPTER 5

application module

`application.`**`analyze`**`()`
Receives uploaded document and comparison document choice and executes logic to compare them.

Returns Information regarding the uploaded document's similarity to regulatory document

Return type html

`application.`**`bubbletest`**`()`
Page for testing

Returns Test page

Return type html

`application.`**`comparisoninfo`**`()`
Comparison Information

Returns graph html page that describes the Linguistic Analyzer's Comparison Score

Return type html

`application.`**`getapplicationconfig`**`()`

Returns json application config file

Returns applicationconfig.json

Return type json file

`application.`**`getbackgroundimg`**`()`
Returns png image of file at

Returns graph

Return type png

`application.`**`getbackgroundwordsimg`**`()`
Returns png image of a graph of words background

Returns graph

Return type png

`application.`**`getcsvkeywords`**`()`

Returns csvkeywords.csv

Returns csvkeywords keyword file

Return type csv

`application.`**`getdocumentationhome`**`()`

Returns index page nested in Documentation/_build/html which is the home page for our Sphinx-generated documentation

Returns html text

`application.`**`getkwfreeqimage`**`()`

Returns Keyword frequency graph

Returns graph

Return type png

`application.`**`getkwsalienceimage`**`()`

Returns png image of a graph of top salience keywords

Returns graph

Return type png

`application.`**`getkwscoresimage`**`()`

Returns png image of a graph of keyword scores

Returns graph

Return type png

`application.`**`getlinguisticanalyzerlog`**`()`

Returns LinguisticAnalyzer.log

Returns log file

Return type log

`application.`**`getregdockws`**`()`

Returns Reg_Keywords.txt

Returns regulatory doc keyword file

Return type txt

`application.`**`gettestkeywords`**`()`

Returns test_keywords.csv

Returns test_keywords doc keyword file

Return type csv

`application.`**`getuserdockws`**`()`

Returns Keywords.txt

Returns keyword file

Return type txt

`application.`**`indexjs`**`()`

Page for testing

Returns Test page

Return type html

`application.`**`keywordbubblechart`**`()`
Returns bubble chart html page

Returns bubble chart html page

Return type html

`application.`**`main`**`()`
Home page of the Linguistic Analyzer API

Returns Home page

Return type html

`application.`**`newregdoc`**`()`
Adds new regulatory document

Returns none

Return type none

`application.`**`project`**`()`
Returns an html page containing details about the Linguistic Analyzer project.

Returns Home page

Return type html

`application.`**`resource_path`**`(`*relative_path*`)`
Summary: Function to determine correct file path of directories for use within an IDE or executable.

Parameters **`relative_path`** (*`str`*) – the path of a directory relative to a local environment

Returns base_path in relation to executable environment and relative_path of local environment

Return type string

`application.`**`reusablebubble`**`()`
Page for testing

Returns Test page

Return type html

`application.`**`reusablebubblejs`**`()`
Page for testing

Returns Test page

Return type html

`application.`**`yulesinfo`**`()`
Yule’s Info

Returns Page that describes Yule’s k and Yule’s i algorithms

Return type html

CHAPTER 6

unit_tests package

6.1 test_analyze module

class `unit_tests.test_analyze.`**`TestAnalyze`**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

`test_analyze`()
Summary: Tests the Analyze() function

6.2 test_extractmicrosoftdocxtext module

class `unit_tests.test_extractmicrosoftdocxtext.`**`TestExtractmicrosoftdocxtext`**(*methodName='runTe*
Bases: `unittest.case.TestCase`

`test_extractmicrosoftdocxtext`()
Summary: Tests the extractmicrosoftdoctet() function

6.3 test_extractpdftext module

class `unit_tests.test_extractpdftext.`**`TestExtractpdftext`**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

`test_extractpdftext`()
Summary: Tests the extractpdftext() function

6.4 test_outputkeywordtotext module

class `unit_tests.test_outputkeywordtotext.`**`TestOutputkeywordtotext`**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

test_outputkeywordtotext()

6.5 test_pdfanddocxarereadthesame module

class unit_tests.test_pdfanddocxarereadthesame.**TestEnsurepdfanddocxarereadthesame**(*methodNam*

Bases: unittest.case.TestCase

test_ensurepdfanddocarereadthesame()

Summary: tests whether extractpdftext() and extractdocxtext() return the same exact information when given the same document in different formats

CHAPTER 7

behave_tests package

7.1 tutorial module

`behave_tests.tutorial.steps.tutorial.`**`step_impl`**`(`*context*`)`

@type context: behave.runner.Context

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index