CLACK Reference Manual

Kevin Shrestha

18/11/2018

Contents

1 Overview **1**
1.1 Example of the algorithm 1

2 JSON Input **2**
2.2 Example 3

3 Functions **4**
3.1 `Set_KWIC_Size()` 4
3.2 `Set_KWIC_Prefix_Remove_Regex()` and `Set_KWIC_Suffix_Remove_Regex()` 5
3.3 `Get_Occurences()` 5
3.4 `Check_Keyword()` 6
3.5 `Classify_Text()` 6

1 Overview

A comprehensive text mining tool used to classify text corpus by identifying keywords present and analysing its context. Fuzzy text matching identifies keywords, accounting for potential misspellings, and proceeds to highlight words prefixed and suffixed by the keyword to understand its context. The main functionality of the tool can be summarised in three steps and are as follows:

1. **Identify Keyword:** The first step of the process is to identify keywords that are present in a given text corpus of interest. Agrep is utilised to identify the occurrence of keywords. This allows for approximate matching to ensure misspellings of keywords are accounted for to some degree depending on the quality of the text corpus. Sensitivity of the fuzzy text matching can be declared for each keyword discretely in the JSON file (see JSON input section)
2. **Contextualise Keyword:** Once position of keywords are identified, associated prefix and suffix words is extracted. This is extracted using the KWIC tool from the Quanteda package.
3. **Classify Corpus:** Extracted keywords with prefix and suffix words are then categorised by classifications identified in the JSON file (see JSON input section).

1.1 Example of the algorithm

The following example provides a run through of the algorithm on the electronic medical record (EMR) data. The purpose of classification in this case is to classify patient's likelyhood of having a heart attack by identifying symptoms of Acute Coronary Syndrome (ACS) related chest pain from the patient's Emergency Department (ED) Presenting Information. By reviewing the the context of how the symptom is present, the algorithm classifies the likelyhood from high to low chance of heart attack. The following example table lists keywords being searched in the ED Presenting Information corpus and associated descriptions with its classification.

Keyword	Context Descriptions	Classification	Comment
chest pain	left sided	HIGH	left sided chest pain is considered to be highly correlated to heart attack
chest pain	pleuritic	LOW	chest pain, pleuritic in nature is more inclined to be respiratory related symptom
chest pain	intermittent	MID	chest pain described as intermittent has a possibility of being ACS related
chest tightness		HIGH	any forms of chest tightness is classified as HIGH

The classification above is applied to the following text corpus a patient's emergency presenting information:

"Left sided chest pain radiating to jaw, heavy and sharp in nature, nausa, mild SOB. Aspirin 300mgs, 900mcg S/L GTN, pain initially 6 now 3"

Step 1: Identify Keyword
Keywords from the table above are being identified in the following sample corpus:

*"Left sided **chest pain** radiating to jaw, heavy and sharp in nature, nausa, mild SOB. Aspirin 300mgs, 900mcg S/L GTN, pain initially 6 now 3"*
Here we see "chest pain" being identified.

Step 2: Contextualise Keyword
Prefix and suffix words associated with the keyword identified are extracted, search limited to 3 words for both prefix and suffix:

*"**Left sided** chest pain **radiating to jaw**, heavy and sharp in nature, nausa, mild SOB. Aspirin 300mgs, 900mcg S/L GTN, pain initially 6 now 3"*
"left sided" is being extracted as the prefix and "radiating to jaw" for suffix.

Step 3: Classify Corpus
The contextual words extracted are then analysed to check whether they match any terms from the Context Description column in the table above. Here we see that in the prefix "left sided" is one of terms in the Context Description which is classified to be HIGH. Hence the corpus is classed as **HIGH** chance of being ACS related.

2 JSON Input

A JSON file is used as the input to this tool. It contains parameters such as the keywords being searched, prefix/suffix terms being analysed and the sensitivity of fuzzy text matching. The format of the JSON file is as follows:

```
[
{"title" : "string",
"keywords" : ["list:string"],
"approx.match" : ["list:numerical"],
"false.positive.matches" : ["list:string"],
"prefix": ["list:string"],
"suffix": ["list:string"],
```

```
"prefix.approx.match" : ["list:numerical"],
"suffix.approx.match" : ["list:numerical"],
"prefix.descriptions" : ["list:string"],
"suffix.descriptions" : ["list:string"],
"prefix.class" : ["list:string"],
"suffix.class" : ["list:string"],
"ignore.prefix": ["list:string"],
"ignore.suffix": ["list:string"],
"ignore.prefix.approx.match" : ["list:numerical"],
"ignore.suffix.approx.match" : ["list:numerical"]
}
]
```

Line 2: Name of the keyword being searched

Line 3: List of different interpretation for the keyword. Eg. "chest pain" can appear abbreviated as "cp"

Line 4: List of sensitivity levels of the fuzzy text matching for each item in **Line 3** respectively.. Value is passed to max.distance parameter of `agrep()`. Length of this list should be the same as list on **Line 3**.

Line 5: Any false positive matches that is to be explicitly ignored. Can occur if sensitivity value of the fuzzy text matching is high.

Line 6: List of prefix words to look out for in relation to the keyword. Refering to example table above, this is where the values of the column Context Descriptions is stored. For example, if we are checking "left sided" preceeding the keyword "chest pain", we inlist the context description, "left sided" in this variable.

Line 7: List of suffix words to look out for in relation to the keyword. Same concept as **Line 6** but checking for Context Descriptions as a suffix to the keyword.

Line 8: List of sensitivity levels of the fuzzy text matching for each item in **Line 6** respectively. Value is passed to max.distance parameter of `agrep()`. Length of this list should be the same as list on **Line 6**.

Line 9: List of sensitivity levels of the fuzzy text matching for each item in **Line 7** respectively.. Value is passed to max.distance parameter of `agrep()`. Length of this list should be the same as list on **Line 7**.

Line 10: List of descriptions for each item in **Line 6** respectively. The description will be stored in the CLASSIFICATION_REASON column if the prefix along with the keyword is found in the text corpus. Length of this list should be the same as list on **Line 6**.

Line 11: List of descriptions for each item in **Line 7** respectively. The description will be stored in the CLASSIFICATION_REASON column if the suffix along with the keyword is found in the text corpus. Length of this list should be the same as list on **Line 7**.

Line 12: List of classfication for each item in **Line 6** respectively. The classification will be used in the CLASSIFICATION column if the prefix along with the keyword is found in the text corpus. Length of this list should be the same as list on **Line 6**.

Line 13: List of classfication for each item in **Line 7** respectively. The classification will be used in the CLASSIFICATION column if the prefix along with the keyword is found in the text corpus. Length of this list should be the same as list on **Line 7**.

Line 14: List any words found in prefix words which should negate the occurence of the keyword. For example, if "no chest pain" is found in text corpus and we would like to ignore this occurence of chest pain because its negated.

Line 15: List any words found in suffix words which should negate the occurence of the keyword. For example, if "chest pain not observed" is found in text corpus and we would like to ignore this occurence of chest pain because its negated.

Line 16: List of sensitivity levels of the fuzzy text matching for each item in **Line 14** respectively. Value is passed to max.distance parameter of `agrep()`. Length of this list should be the same as list on **Line 14**.

Line 17: List of sensitivity levels of the fuzzy text matching for each item in **Line 15** respectively. Value is passed to max.distance parameter of `agrep()`. Length of this list should be the same as list on **Line 15**.

2.2 Example

The following is an example of a JSON input file containing two keywords to be seached for in the text corpus. The keywords being, "chest pain" and "vomiting".

In terms of chest pain, we are searching for keyword strings, "chest pain" and "cp". "chest pain" having `agrep` max.distance sensitivity of 0.1 and "cp" having 0.0 (exact match). Once keywords are identified, we check the context of the keyword and we are looking for chest pain which is "left sided", "plueretic" or "radiating". If any of these are found, its description will be added to the CLASSIFICATION_REASON field of the text corpus and its class to CLASSIFICATION field. Here we ignore cases where "no chest pain", "nil chest pain" or "chest pain denied" is found.

In terms of vomiting, we ignore cases where fuzzy text matching might approximately match "admiting" and "committing". It should be noted that for this keyword we are searching for an empty string as prefix to the keyword. This suggests that we are not concerned about assessing any contextual information of the keyword, we are only concerned about the existence of the keyword for classification.

```
[
{"title" : "chest pain",
"keywords" : ["chest pain", "cp"],
"approx.match" : [0.1, 0.0],
"false.positive.matches" : [],
"prefix": ["left sided", "pleuretic"],
"suffix": ["radiating"],
"prefix.approx.match" : [0.1, 0.1],
"suffix.approx.match" : [0.1],
"prefix.descriptions" : ["left sided cp", "pleuretic cp"],
"suffix.descriptions" : ["radiating cp"],
"prefix.class" : ["HIGH", "LOW"],
"suffix.class" : ["HIGH"],
"ignore.prefix": ["no", "nil"],
"ignore.suffix": ["denied"],
"ignore.prefix.approx.match" : [0.0, 0.0],
"ignore.suffix.approx.match" : [0.1]
},
{"title" : "vomiting",
"keywords" : ["vomiting"],
"approx.match" : [0.1],
"false.positive.matches" : ["admiting","committing"],
"prefix": [""],
"suffix": [],
"prefix.approx.match" : [0.0],
"suffix.approx.match" : [],
"prefix.descriptions" : ["vomiting"],
"suffix.descriptions" : [],
"prefix.class" : ["MID"],
"suffix.class" : [],
"ignore.prefix": ["no", "nil"],
"ignore.suffix": [],
```

```
"ignore.prefix.approx.match" : [0.0, 0.0],
"ignore.suffix.approx.match" : []
}

]
```

3 Functions

3.1 `Set_KWIC_Size()`

Sets the number of prefix/suffix words to extract while analysing keyword in context. By default the value is set as 4. This means 4 word preceeding and 4 words following the keyword is accessed.

For Example ("chest pain" is the keyword):

"PT PRESENTS WITH CHEST PAIN AND LEFT SHOULDER PAIN WITH ASSOC PALP"

"**PT PRESENTS WITH** CHEST PAIN **AND LEFT SHOULDER PAIN** WITH ASSOC PALP"

The function used to extract context of keywords is `kwic()` from the `quanteda` package. The enviornment variable set by this function is passed to the "window" argument of the `kwic()` function.

Arguments:

size: number of words to be extracted for each prefix and suffix

Pseudocode:

```
if input parameter is an integer and is greater than 0
  set the KWIC size to input parameter
else return an error
```

3.2 `Set_KWIC_Prefix_Remove_Regex()` and `Set_KWIC_Suffix_Remove_Regex()`

`Set_KWIC_Prefix_Remove_Regex()`: Sets the `env.kwic.regex.remove.pre` variable. Should store a regex string expression. After prefix words are extracted, data clensing can be performed before analysing the prefix words.

For example, if the algorithm extracts 4 prefix words associated with the keyword "chest pain" in the following example:

*"No nausea. Sharp **chest pain** radiating to jaw. Aspirin 300mgs, 900mcg S/L GTN, pain initially 6 now 3"*

This would extract the following:

"No nausea. Sharp"

Observing this, we see that within the prefix words extracted, the only relevant prefix words associated with the keyword "chest pain" is "Sharp", as "No nausea" is part of another sentence. The `env.kwic.regex.remove.pre` variable serves the purpose of removing these words which may be irrelevant to the keyword by removing them using `gsub`. The regex string is provided to the "pattern" argument in `gsub` in `Check_Keyword()`. By default the value of `env.kwic.regex.remove.pre` is ".*\." , when provided to gsub, removes all words prior to a full stop. This ensures that in cases like one described above, only relevant prefix words which are part of the same sentence as the keyword is being analysed.

The idea behind this is that we can handle this kind of clensing more flexibly, cater for different scenarios and type of data being handled. For instance, the requirement maybe that we need to remove everything before a hyphen instead or any other punctuation, this will allow for that.

`Set_KWIC_Suffix_Remove_Regex()`: Same concept as `Set_KWIC_Prefix_Remove_Regex()` but instead cleanses extracted suffix words. By default the regex string expression is "\..*". This will remove any words after a full stop.

Arguments:

regex.string: regex string expression to be used

3.3 `Get_Occurences()`

Returns a list of all the perturbations of a keyword found in a text corpus.

Arguments:

text: text corpus to be analysed

word: keyword to be search for

distance: sensitivity of the fuzzy text matching. See max.distance parameter for agrep for more information

Pseudocode:

```
text <- convert text variable to string if vector is provided as input
final_list <- declare list variable which will store all different perturbations
              of the keyword present in the text corpus
while new perturbation was found
  Extract a new perturbation of the keyword from the variable, text, using fuzzy text search
  temp.list <- save the perturbation extracted to this variable
  if temp.list is blank (no new perturbation of keyword found) then
    Stop looking for new perturbation, exit while loop
  else
    Add the perturbation of the keyword found to the final_list
    Remove all occurence of value stored in temp.list from text variable
    (so same perturbation is not picked up again)
return final_list
```

3.4 `Check_Keyword()`

Given a keyword and a list of classification associated to the keyword, check if corpus can be classfied. Return list of classification found.

Arguments:

text: text corpus to be analysed

checking.df: keyword with its list of classification provided as a dataframe

Pseudocode:

```
text <- convert text argument variable to lower case
Extract and store information about keyword from the json input data
Check if json input data contains any errors
checklist <- Extract and store the list of keyword perturbations in the text corpus
checklist <- Remove any false positive matches from the list of perturbations
for each keyword from the filtered list of perturbations, checklist
  Prepare keyword to be used with kwic function (remove special character such as "[" and "]")
  kwic_output <- search text corpus for keyword, extract keyword with associated prefix and suffix words
  if keyword occurences are found
    for each occurence of the keyword in the text corpus
```

```
        Cleanse prefix words (see Set_KWIC_Prefix_Remove_Regex())
        Cleanse suffix words (see Set_KWIC_Suffix_Remove_Regex())
        if occurence of keyword is to be negated/ignored (prefix)
          ignore current occurence and check next occurence, skip current iteration
        if occurence of keyword is to be negated/ignored (suffix)
          ignore current occurence and check next occurence, skip current iteration
        if prefix words extracted matches classification declared in the json file
          save the classification matched to output.list
        if suffix words extracted matches classification declared in the json file
          save the classification matched to output.list
return output.list
```

3.5 Classify_Text()

Function called to perform the classification of the text corpuses. It creates two columns in the dataframe provided. CLASSIFICATION column contains all classification that has been identified for the text corpus. CLASSIFICATION_REASON stores the description for the classification.

Arguments:

text.df: dataframe that contains all text corpus being classified

column.name: name of the column that contains the text corpuses in text.df

json.file.name: file name containing the JSON data

Pseudocode:

```
Convert column containing text corpus to be classified to type character
Add CLASSIFICATION and CLASSIFICATION_REASON column to text.df
Read JSON input file
for each text corpus to be classified
  for each keyword declared in the JSON file
    Encode text corpus to UTF-8
    Check keyword in text corpus
    listings <- Store all classifications found in text corpus
Populate CLASSIFICATION and CLASSIFICATION_REASON column with values from variable, listings
return classified text corpuses dataframe, text.df
```