Text prediction app documentation - Functions

Ivy Woo

18 February 2021

This is a documentation on all self-defined functions used in building the text prediction app.

All functions are either fully or partially provided in the functions.R file in the app's Github repo. Several selected functions have some parts removed so as to prevent the code being misused, e.g., plagiarized by dishonest participants of the same project course.

```
makeToken(data)
```

This function uses a pipe of the tokenizers library functions and R base functions to tokenize the raw text data into our desired form (refer to the dataset documentation for information in this regard). The pipeline does the following step by step:

1. To split the texts into sentences. (So as to distinguish sentence-begins for carrying out point 3 below.)
2. To further split each sentence into tokens of single words and transform all texts into lower case at the same time.
3. To add a sentence-begin tag <s> to the beginning of each sentence.
4. To remove all tokens which are not English alphabets nor punctuations that are supported (comma or fullstop).

Argument(s)

data: the raw text data.

```
junk(token, percentage = 0.9, monitor = TRUE)
```

This function takes in the tokens and returns a character vector which contains elements which are classified as junk. Non-junk is defined as the following: all the most frequently appearing words in the tokens which constitute certain threshold percentage of the total token counts.

Argument(s)

token: a character vector in which the elements are the tokens. (Here the output from the makeToken function.)
percentage: the threshold percentage to determine junk/non-junk words.
monitor: if TRUE, the processed percentage in each sub-step (in total three) of the computation will be printed out during the function's run.

```
makeTokUnk(token, percentage = 0.9)
```

This function takes in the tokens and returns the set of tokens where all junk words are replaced by the unknown token <unk>.

This function wraps the junk function.

Argument(s)

token: a character vector in which the elements are the tokens. (Here the output from the makeToken function.)
percentage: the threshold percentage to determine junk/non-junk words.

```
splitTok(token, nsplit = 10)
```

This function splits the given token vector into smaller objects. Each split object has approximately equal number of sentences. This is useful when the token size is large and hinders the subsequent computations.

Argument(s)

token: a character vector in which the elements are the tokens. (Here the output from the makeTokUnk function.)
nsplit: how many objects the token vector is to be split into.

```
buildDict(h, token, ngram = 4, monitor = TRUE)
```

This function takes in a dictionary object defined by the collections package, the tokens, and returns a dictionary tree (refer to the model-construction documentation for a thorough explanation of the structure).

Argument(s)

h: a dictionary object defined by the collections package.
token: a character vector in which the elements are the tokens. (Here output from the makeTokUnk function.)
ngram: the number n in n-gram, specifying dictionary tree of the n-gram model to be built.

monitor: if TRUE, the computation progress (in form of the processed percentage) will be printed out during the function's run.

sync(h, que, ngram)

This is a function wrapped in the buildDict function for syncing the progress of writing in different n-grams as the tokens are scanned through.

Argument(s)

h: a dictionary object defined by the collections package.
que: a queue object defined by the collections package.
ngram: the number n in n-gram, specifying dictionary tree of the n-gram model to be built.

increment(h, quelist)

This is a function wrapped in the buildDict function for writing the n-grams into the dictionary tree.

Argument(s)

h: a dictionary object defined by the collections package.
quelist: the elements in a queue object defined by the collections package as list.

recommendation(h, nrec = −1, monitor = TRUE)

This function builds the recommendation list into the dictionary tree. This is needed when the dictionary tree is used directly for looking up predictions (i.e. not building a subsequent data table).

Argument(s)

node: a dictionary tree. (Here the output from the buildDict function.)
nrec: The number of predictions to be retained in the tree. If negative, all existing predictions are retained.
monitor: if TRUE, the computation progress (in form of the processed percentage) will be printed out during the function's run.

buildDT(h, nrec = −1, mincnt = 1, monitor = TRUE)

This function builds a two-column data table (defined by the data.table package) from a dictionary tree. Column 1 contains the n-grams with the last word removed; column 2 contains the top k predictions for the string in column 1 in the respective row, which are concatenated as one string.

Argument(s)

h: a dictionary tree. (Here the output from the buildDict function.)
nrec: The number of predictions k to be retained in the resulting data table. If negative, all existing predictions are retained.
mincnt: the minimum count for an n-gram to be included into the data table.
monitor: if TRUE, the computation progress (in form of the processed percentage) will be printed out during the function's run.

```
cntgrams(h)
```

This function returns the total number of n-grams in a dictionary tree.

Argument(s)

h: a dictionary tree.

```
lookup(DT, path, ngram = 4, nxt = 5)
```

This is the function for looking up the prediction(s) given some texts.

Argument(s)

DT: a data table object defined by the data.table package. (Here the output from the buildDT function.)
path: the string of which the next word is to be predicted.
ngram: the maximum n in n-grams available from the data table.
nxt: number of predictions of the next word to be output (dependent on the information available in the data table).

```
splitInput(string)
```

This function splits some given string into vector of characters, which could then be processed by the lookup function.

This function is wrapped in the lookup function.

Argument(s)

string: a single-element character vector.

```
searchDT(output, DT, path)
```

This function serves to iteratively look up the data table to collect predictions until the required number is attained. It is wrapped in the lookup function.

Argument(s)

output: self-defined object (inside the lookup function) which acts as a container to memorize the predictions as the function iteratively looks up the data table.
DT: a data table object defined by the data.table package. (Here the output from the buildDT function.)
path: the string of which the next word is to be predicted.