classecol: vignette

Thomas Frederick Johnson Github: GitTFJ

26th November 2020

classecol

classecol is a package to perform nature-related text classifications of public opinion data (trained using twitter data). classecol provides a series of functions which can clean data, pull together outputs from multiple sentiment analysis approaches into one function, and has three models to classify text: bio - to assess who the user is e.g. a person, expert, nature organisation, or something else; nature - to assess if the tweet is relevant to nature (or the natural world), and if so, describe whether the tweet is showing concern, interest or fear; hunting - to assess if the tweet is relevant to hunting, and if so, describe whether the tweet is pro- or against-hunting.

Instructions to prepare package

This package is run through R but is reliant on a python back-end (which dramitcally improves the speed of the classification). So before running any code, you will need to install a version of python - we recommend Python 3.6.X which is what the package has been extensively tested on; available at (https://www.python.org/downloads/release/python-369/). You will also need to install a selection of packages on python: keras, tensorflow, pandas, nltk, bs4, sklearn, scikit-learn (version 0.19.1). Packages can be installed following (https://packaging.python.org/tutorials/installing-packages/). If at anypoint the error 'Module not found' appears, install the listed module/package following the above instructions. If you are new to python, or come across any issues in the install, we would recommend watching the following tutorial (https://youtu.be/MNPp_18nhGk).

The most recent and throughly tested version of classecol is only available as a github repository so needs to be installed through github

```
library(devtools)
install_github("GitTFJ/classecol")
library(classecol)
library(reticulate)
```

The python text classification models which classecol is reliant on are not automatically downloaded when the package is installed. Instead, its necassary to download and save an additional github repository, which is automated through the 'download_models' function. To encuorage a seamless use of classecol, we recommend storing these models alongside the classecol package, which can found locally using the 'find.package' function.

```
direc = paste(find.package("classecol"),"/models", sep = "")
download_models(direc)
```

'reticulate' offers a function to link the R classecol package to the python backend which contains the classification models. However, this automated function can perform inconsistantly, so recommend manuallly specifying pythons absolute filepath location. The file to search for is 'python.exe'. You will also need to specify the location you have downloaded the models to and then send this location to python with the function 'r_to_py'

```
reticulate::use_python("C:/Users/mn826766/AppData/Local/Continuum/anaconda3/python.exe")
#Specify your own path -search for python.exe within your file explorer
direc = paste(direc, "/classecol-models-master/", sep = "")
model_directory = reticulate::r_to_py(direc)
```

At this point we are ready to prepare and classify the data. It is very important that all naming conventions are matched, otherwise the classification will fail e.g. the text needs to be in a column called 'text', in a dataframe called 'df', and assigned as 'data' using the 'reticulate::r_to_py' function. See below.

Hunting classifier

The hunting classifier 'hun_class()' works best with twitter data after a simple clean. The 'type' parameter within the 'hun_class()' function specifies the type of text classification model, options include: Relevance - use this model when the objective is to identify if text is relevant (or not) to hunting; Stance - use this model on text relevant to hunting, where the objective is to identify the stance of the text (pro- or against-hunting); Full - use this model when the objective is to identify text that is relevant to hunting, and when relevant, describe the stance. Combines the relevance and stance steps into one model.

```
df = data.frame(
  text = c(
    "I hate hunting. Ban it now!",
    "Cant wait to go camping this weekend #hunting #fishing",
    "Hunting for my car keys"),
  stringsAsFactors = F)
df$text = classecol::clean(df$text, level = "simple")
data = reticulate::r_to_py(df)
hun_class(
  type = "Full",
  directory = direc)
```

```
## [1] "Relevant (against-hunting)" "Relevant (pro-hunting)"
## [3] "Irrelevant"
```

Nature classifier

The nature classifier 'nat_class()' works best with twitter data after a full clean and also requires sentiment analysis on the text. The 'type' parameter within the 'nat_class()' function specifies the type of text classification model, options include: Relevance - use this model when the objective is to identify if text is relevant (or not) to nature; Stance - use this model on text relevant to nature, where the objective is to identify the stance of the text (positive- or negative phrasing); Trimmed - use this model when the objective is to identify text that is relevant to nature, and when relevant, describe the stance, combining the relevance and stance steps into one model; Full - use this model when the objective is to identify text that is relevant to hunting, and when relevant, describe the stance. Full differs to the Trimmed model, as the Full model includes the low accuracy 'Against-nature' category.

```
df = data.frame(
  text = c(
    "I love walking in nature - so serene",
    "Why are the government not stopping the destruction of the rainforest?!",
    "Tiger wins the PGA tour again!"),
  stringsAsFactors = F)
df$text = classecol::clean(df$text, level = "full")
sm = as.matrix(cbind(
  valence(df$text),
  lang_eng(df$text),
  senti_matrix(df$text)))
data = reticulate::r_to_py(df)
sent_mat = reticulate::r_to_py(sm)
nat_class(
  type = "Trimmed",
  directory = direc)
```

```
## [1] "Pro-nature (positive phrasing)" "Pro-nature (negative phrasing)"
## [3] "Irrelevant"
```

Bio classifier

The biographical classifier 'bio_class()' works best with twitter data in its raw form, so none of the text should be cleaned. However, it is neccasary to join the twitter name and description into one column named 'text' split with a space. The 'type' parameter within the 'bio_class()' function specifies the type of text classification model, options include: Person - use this model when the objective is to identify whether a user is a person or not; Expert - use this model on users classified as a 'person' to identify whether the person is a nature expert or a member of the general public; Full - use this model to identify whether the user is a person, expert, nature organisation, or other. Full wraps the Person and Expert models into one function, and can also identify nature organisations.

```
df = data.frame(
  name = c(
    "Jane Doe ",
    "Thomas Frederick Johnson",
    "Fictional University"),
  description = c(
  "Business leader, banker, parent, and cyclist",
  "Ecology and conservation researcher",
  "Campus life and study at the Fictional University. Follow for news and updates"),
  stringsAsFactors = F)
df$text = paste(df$name, df$description)
data = reticulate::r_to_py(df)
bio_class(
  type = "Full",
  directory = direc)
```

```
## [1] "Person" "Expert" "Other"
```

Other classecol functions

clean()

Cleans social media text converting up to 1000 multi-word nature-related hashtags and over 1500 abbreviations into readable text. This function also converts nearly 150 emoticons and 175 slang-words into readable text using terms sourced from the lexicon R package (Rinker, 2019a). Can select a 'simple' or 'full' clean, where a 'simple' clean includes: hashtag conversion, removal of twitter specific syntax (e.g. RTs), removal of urls, conversion of emoticons, removal of special characters. 'full' includes all of the above as well as converting abbreviations, slang, grades and ratings. The cleaning process can also be completely customised, selecting or removing any paricular claning element. For example, one could do a 'full' clean, but specify that hashtags should not be converted (see below).

```
text = "BTW tomorrow will be the best #TrophyHunting :)"
text
```

```
## [1] "BTW tomorrow will be the best #TrophyHunting :)"
```

```
clean(text, level = "simple")
```

```
## [1] "BTW tomorrow will be the best trophy hunting smiley"
```

```
clean(text, level = "full")
```

```
## [1] "by the way tomorrow will be the best trophy hunting smiley"
```

```
clean(text, level = "full", hashtag = F)
```

```
## [1] "by the way tomorrow will be the best TrophyHunting smiley"
```

valence()

Checks for the presence of negator (flips the meaning of the text e.g. I am NOT sad), amplifier (adds intensity to the text e.g. I am VERY sad), de-amplifier (softens the text e.g. I am KIND OF sad), and adversative-conjunction terms (overrules the previous sentiment e.g. I am sad, BUT proud), which could alter the meaning of the text. Terms and examples sourced from the lexicon (Rinker, 2019a) and sentiment (Rinker, 2019b) R packages.

```
rbind(
  valence("I am not sad"),
  valence("I am very sad"),
  valence("I am kind of sad"),
  valence("I am sad, but proud"))
```

```
##   negator amplifier deamplifier ad_conjunction
## 1       1         0           0              0
## 2       0         1           0              0
## 3       0         0           1              0
## 4       0         0           0              1
```

contract()

Performs stemming (trims word to their simplest form e.g. cars becomes car) and lemmatisation (identifies the core theme of a term, bringing synonymous terms into one word e.g. automobile becomes car) within R. Function is a wrapper for textstem (Rinker, 2018), qdap (Rinker, 2020), and lexicon (Rinker, 2019a) R packages. This function is not necessary when running any of the hun_class, nat_class, and bio_class models, as the stemming and lemmatisation are conducted within the substantially faster Python program.

```
contract("consulting")
```

```
## [1] " consult "
```

lang_eng()

Checks if the language is English or not, the hun_class, nat_class, and bio_class models are designed for English text. Function is a wrapper to the cld2 (Ooms & Sites, 2018) R package.

```
lang_eng("hallo und willkommen bei classecol")#German
```

```
## [1] 0
```

```
lang_eng("hello and welcome to classecol")#English
```

```
## [1] 1
```

senti_matrix()

Conducts sentiment analysis (assesses polarity of text) using 11 approaches and pulls all approaches into a matrix of sentiment. Approaches are drawn from the sentimentr (Rinker, 2019b), lexicon (Rinker, 2019a), syuzhet (Jockers, 2017), and meanr (Schmidt, 2019) R packages.

```
rbind(
  senti_matrix("I love wildlife so much, This is the best day ever"),
  senti_matrix("I hate wildlife so much, This is the worst day ever"))
```

```
##   jockers_rinker jockers huliu loughran_mcdonald senticnet sentiword
## 1           0.56    0.56  0.84              0.3      0.37      0.42
## 2          -0.56   -0.56 -0.84             -0.3     -0.53     -0.33
##   socal_google nrc afinn bing meanr
## 1         1.49   1     6    2     2
## 2         0.00  -1    -6   -2    -2
```