Web Extracting of Emotions

This program uses Google Youtube API to retrieve information about videos, and their comments, in order to offer various ways to explore Youtube. In doing so, you will be able to get videos based on criteria which can be with either some prior knowledge or in a random fashion. For example, it is possible to start from list of terms that will be used to search for videos. You can also supply a list of videos that will be used as a batch to further explore related videos. Please refer to this documentation to obtain more details about our program's capability. The following sections gives insight into the program's implementaiton.

Architecture design

The program is written in Python and consists of 6 files (and 2 example files).

Class youtube_api (from class_api.py)

This class implements methods to query the Youtube API (see documentation here). You can create an instance of this class this way :

```
from class_api import youtube_api
youtube_key = "AIzaSyCxjD8L9yI6yRcFPNZ7brKMFzQhOWNOzIs"
ya = youtube_api(youtube_key)
```

Methods :

```
def request_related_videos(self, vid, order, number_video, location_settings):
  ...
```

Return a list of *number_video* video IDs related to a specific video *vid*.

```
def request_termed_videos(self, order, list_terms, number_video, location_settings):
  ...
```

Return a list of *number_video* video IDs that are matching a list of terms (as a simple search query would do).

```
def request_video_information(self, vid):
  ...
```

Return data about video *vid* and stores these informations in the database.

```
def request_video_comments(self, vid):
  ...
```

Return all comments from video *vid* and stores them in the database.

The methods **request_video_information** and **request_video_comments** include decorators @check_videos_db and @check_comments_db. They are function wrappers that check whether *vid* already exists in the database. If it's not the case, a request is made to Youtube API. These decorators can be disabled easily with comments, this way no data is stored in the database and data always come from Youtube API.

Parameters :

- *vid* is a video ID (ex : pNjLMFrQAvw).
- *number_video* indicates the number of video IDs that will be returned.
- *order* is a parameter that specifies how to sort results from the request, several are possible : relevance, date, viewCount, rating, title (sorted alphabetically), videoCount (sorted in descending order of their number of uploaded videos).
- *location_settings* parameter can be omitted because very few videos embedded such information on Youtube.

Class youtube_database (from database.py)

This class includes all basic methods to create the database and interact with it : insertion and retrieval on two tables "videos" and "comments". Sqlite database doesn't allow UTF-8 format text to be stored, so it requires to encode UTF-8 strings into python string representation (*str* type) via **encode("utf-8")** function. The **encode("utf-8")** is used direcly inside methods of **youtube_api** class when raw data is retrieved from Youtube.

Class yandex (from class_api.py)

This class implements methods to use Yandex API that allows to translate words in many languages (see documentation here). Yandex supports more than 80 languages and is free to use (under certain limits).

```
from class_api import yandex
yandex_key = "trnsl.1.1.20190107T142924Z.19d76f10aaa3f91a"
yx = yandex(yandex_key)
```

Methods

```
def translate(self, text, to_lang):
  ...
```

One can use this method to translate a *text* into a specific language given by *to_lang*. Return a list of strings, where each entry reflects a possible translation of *text* or an empty list if no translation is available.

```
def build_wordlist(self, wordlist_en, lang_support):
  ...
```

This method creates and returns a list of words *wordlist* (that is used to identify words in comments). This list is based on *base_wordlist_en* where all words are translated into several languages given by *lang_support*.

Parameters :

- *text* is a text which must be written in english (either a word or a sentence).
- *to_lang* is a 2-length string code (for example "fr") representing the target language for the translation.
- *wordlist_en* is a list of english words.
- *lang_support* is a list of *to_lang* code languages.
- *wordlist* is a list of words (translated in different languages)

Class word_analysis (from word_analysis.py)

The goal of this class is to provide methods to process and analyze content coming from Youtube comments. In order to work properly, it requieres to install beforehand nltk and nltk.corpus (stopwords, brown) python libraries.

```
from word_analysis import word_analysis
wa = word_analysis()
```

Methods

```
def build_corpus(self):
 ...
```

Load in memory all relevant words from "brown" corpus.

```
def get_random_word(self):
 ...
```

Extract randomly a word previously loading from "brown" corpus and returns it. This method is used by **explore_youtube_randomly** (see below, class client_api).

```
def get_decomposed_text(self, text, nb_words, mininum_size):
  ...
```

Decompose *text* into sequence of words of length exactly equals to *nb_words* and having a size greater than *minimum_size*.

```
def get_comments_nb_matching(self, comments, wordlist):
  ...
```

Calculate the proportion of comments having at least one word in common with *wordlist* and returns it. Calculate the frequency distribution of terms (from *wordlist*) matching in the comments as well as its total number.

Class client_api (from client_api.py)

This class depends of two classes youtube_api and word_analysis which are declared when client_api is instanciated.

```
from client_api import client_api
youtube_key = "AIzaSyCxjD8L9yI6yRcFPNZ7brKMFzQhOWNOzIs"
ca = client_api(youtube_key)
```

The collection of methods here allows to explore Youtube in various way.

Methods :

```
def explore_youtube(self, filename_net, list_vid, depth, number_videos,
order_criteria = "relevance", location_settings = None):
  ...
```

Explore Youtube from a list of video IDs *list_vid*. Throughout the process one video can lead to multiple other videos and so on, so that it creates a network of videos which is stored in a file named *filename_net*. Return *number_videos* of video IDs explored this way (containing also all video IDs of *list_vid*).

```
def explore_youtube_by_wordlist(self, filename_net, list_vid, depth, number_videos,
nb_best_matching, order_criteria = "relevance", location_settings = None):
  ...
```

Explore Youtube in the same way as does the previous method but this time only the first *nb_best_matching* best related videos are selected. This selection is based on how many words, in the comments of a video, are similar to the words from *list_words*. It also creates a network out of it.

```
def explore_youtube_randomly(self, number_hits):
  ...
```

A set of words is randomly collected from a corpus in order to query Youtube. A list of 15 video IDs is retrieved for each word where only one is randomly selected. In this way, it returns a list of *number_hits* video IDs.

Parameters :

- *list_vid* is a list of video IDs that will serve as a basis for exploration.
- *depth* specifies how many times we want to keep on exploring by following lastest retrieved video IDs
- *number_video* indicates the number of video IDs that will be returned.
- *order* is a parameter that specifies how to sort results from the request, several are possible : relevance, date, viewCount, rating, title (sorted alphabetically), videoCount (sorted in descending order of their number of uploaded videos).
- *location_settings* parameter can be omitted because very few videos embedded such information on Youtube.

Usage

Please look at files "**run.py**" and "**analyze.py**" to get more possible usage of the program. There is an example of how you can use the program :

```
 # -*- coding: utf-8 -*-

from client_api import client_api
from class_api import yandex
from word_analysis import word_analysis

youtube_key = "AIzaSyCxjD8L9yI6yRcFPNZ7brKMFzQhOWNOzIs"
ca = client_api(youtube_key)

yandex_key = "trnsl.1.1.20190107T142924Z.19d76f10aaa3f91a"
yx = yandex(yandex_key)

wordlist_english = ["frisson", "frissons", "chill", "chills",
"gooseflesh", "goosebump", "goosebumps"]
lang_support = ["fr", "es", "it"]

wordlist = yx.build_wordlist(wordlist_english, lang_support)
for w in wordlist:
    print w
```

```
pelle d'oca
escalofríos
la chair de poule
```

```
frisson
enfriar
freddo
goosebumps
des frissons
la pelle d'oca
goosebump
gooseflesh
la piel de gallina
frissons
brividi
piel de gallina
chills
brivido
chill
```

```
# searching terms
list_terms = ["frisson", "gooseflesh"]
# retrieve 5 video IDs that match "frisson" or "goosflesh"
list_vid = ca.get_list_video_by_termlist(list_terms, 5)
print list_vid
```

```
[u'eJrkGK58OOo', u'hBDsI6YfpT0', u'9R0Hud17A2k', u'g2KoewmFGlc', u'DS5ZYNjL3XI']
```

```
wa = word_analysis()

for vid in list_vid:
    comments = ca.ya.request_video_comments(vid)
    results = wa.get_comments_nb_matching(comments, wordlist)
    (data_wordlist, nb_matching, matching_ratio) = results
    print vid, matching_ratio
```

```
eJrkGK58OOo 0.140350877193
hBDsI6YfpT0 0.07
9R0Hud17A2k 0.0284090909091
g2KoewmFGlc 0.195075757576
DS5ZYNjL3XI 0.00607287449393
```