YouTube comment sentiment analysis with web interface

Søren Howe Gersager
s094557
Technical University of Denmark

Anders Lønberg Rahbek
s107029
Technical University of Denmark

***Abstract*—We have created a webservice that performs sentiment analysis on YouTube videos using the Google Data APIs, by using a self combined corpus and a trained simple machine learning classifier**

I. Introduction

This paper showcases the results of a project, analyzing YouTube comments using Python.
It describes the methods and techniques used to create the analysis and discusses the paths taken as well as further potential work to be done.

II. Methods

We have divided the project into classes specific for the different areas our project encompasses: a youtube scraper, a sentiment analyser and a webservice.

A. Webservice

For the webservice we use the Flask [1] using the Jinja2 [2] templating engine, this allows us to separate the code from the markup.

We also used Javascript and SASS [3], to a lesser degree for the frontend development of the webservice.

While developing we have used the built-in flask web server for ease of use, for deployment however, a traditional web server like Apache with mod_fastcgi would probably be the better option for its performance and security.

B. Sentiment analyser

The sentiment analyser is based on the NLTK package [4]. We train a Naive Bayes classifier and saves the classifier as a pickle file, so we next time don't have to train, but simply load the classifier into NLTK.
The classifier is trained based on a combined corpus of tweets from Twitter [5] and movie reviews [6].

The corpus has been pre-prossesed so there is equal number of positive and negative texts - 13333 each. The hashtags and user-specified tags ('@'-tag) has also been removed from the twitter corpus. Reversed, there has been added a few emoticons like ':D' and ':(' (positive and negative respectively).

The sentiment analysis itself, is done by a method which takes the comments as input and do sentiment analysis on all of them. Then it's finding a overall classification of the video by making a majority vote of the comments sentiment. The classification can be one of five possibillities: Strong negative, slight negative, neutral, slight positive or strong positive.

C. YouTube scraper

The YouTube scraper uses the Google Data API's for fetching data from youtube and requests as the library making the HTTP requests. We realize a python gdata module already exists,

D. General

The project runs Python 3 and have not been tested on Python 2, however all external libraries we use has a Python 2 equivalent. The only real hurdle for compatibility would be the difference in the Python Standard Library.

In the beginning of the project we simply used the sqlite [7] module for communication with the database, however we later realized this was cumbersome because we had to manually write functions for inserting all the scraped information into the database, we decided to switch to the sqlalchemy object relational mapper [8] for database communication. By using an ORM we saved time and could do database transactions in an easy way using our specified models.

Behind the sqlalchemy ORM we use the sqlite database, as it allows for fast prototyping, Figure 2 shows an ER diagram for the database schema.

Code listings can be found in the listings section.

III. Results

Figure 1 shows a screenshot of the webservice with a sentiment analysis analysis of a YouTube video.

A. Sentiment analysis

Our sentiment analysis showed an accuracy of 68.58%, which is decent. This result is based on a test-set and by using util.accuracy in the NLTK package.
The training of the classifier is taking some time. This time is way too long if it should train every time (approx. 40 minutes). This is touched upon further in the Discussion section.

B. Code checking

For syntax and static checking we used flake8[9], a wrapper of pyflakes and pep8, we also used pylint[10] and pep257[11]. We integrated these checks as a part of our testing so we could quickly assess if our code was sound. In addition to checking our code we also checked our tests.

C. Testing

For testing we primarily used the python unittest library in combination with nose [12], we used coverage [13] for code coverage making sure we had a test coverage of 100%. As written above we integrated the flake8 and pylint checking into the tests. We used both unit-testing and integration testing in testing the various components in our project. We have divided the tests into several files where each test file corresponds to a module and one for the code checking tests.

D. Profiling

We have not analysed the project by profiling but we estimate the HTTP requests and the classification performed will be the bottleneck. This is touched upon further in the Discussion section.

IV. Discussion

As our sentiment corpus consists of words in english, the sentiment analysis will only work for English texts.

As the webservice makes use of the Google Data APIs it has to make several HTTP GET requests for fetching comments, right now 1 request for every 25 comments, this serves as a bottleneck for analysing YouTube videos with large amounts of comments. Because of this bottleneck it results in a delay for the user the first time a new video is analysed, however subsequent requests on the particular video will fetch the information from the database eliminating the delay. The database fetch happens only if the number of comments for the video are unchanged from the last analysis. Further work could be spent looking into optimizing the HTTP requests made for example by parallelization of the requests. The classification also take up a considerable amount of time and further time could be spent optimizing this as well. Profiling the project could be done using the cProfile module[14].

As mentioned earlier, the training of the classifier is taking too long. As we saves the classifier, so we next time just can load it, it's something we can live with.
By reducing the corpus, we can cut down the training time. But this will affect the accuracy of the classifier. So it's a trade-off issue. As we don't know exactly how much the trade-off is and how much of the corpus we can cut-off before the accuracy of the classifier is decreasing too much, this is an area to look into the next step to go for optimizing the sentiment analysis.

V. Conclusion

We have implemented a YouTube sentiment web service mining YouTube comments and performing sentiment analysis.
By using a simple Naive Bayes classifier and relative big corpus, we obtained a decent accuracy of 68.58%
Due to the nature of the service and restrictions in the Google Data APIs it works best with a small number of comments, however we mediate it somewhat by saving the results to a local database.

References

[1] Flask. [Online]. Available: http://www.flask.pocoo.org/
[2] Jinja2. [Online]. Available: http://jinja.pocoo.org/docs/dev/
[3] SASS. [Online]. Available: http://sass-lang.com/
[4] NLTK. [Online]. Available: http://www.nltk.org/
[5] Twitter corpus. [Online]. Available: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/
[6] Movie reviews. Bo Pang, Cornell University, Ithaca. [Online]. Available: http://www.cs.cornell.edu/people/pabo/movie-review-data/
[7] SQLite. [Online]. Available: http://www.sqlite.org/
[8] SQLAlchemy ORM. [Online]. Available: http://www.sqlalchemy.org/
[9] Flake8. [Online]. Available: http://flake8.readthedocs.org/en/2.2.3/
[10] pylint. [Online]. Available: http://www.logilab.org/857
[11] pep257. [Online]. Available: https://pypi.python.org/pypi/pep257
[12] nose. [Online]. Available: https://nose.readthedocs.org/en/latest/
[13] coverage. [Online]. Available: http://nedbatchelder.com/code/coverage/
[14] cProfile. [Online]. Available: https://docs.python.org/3/library/profile.html

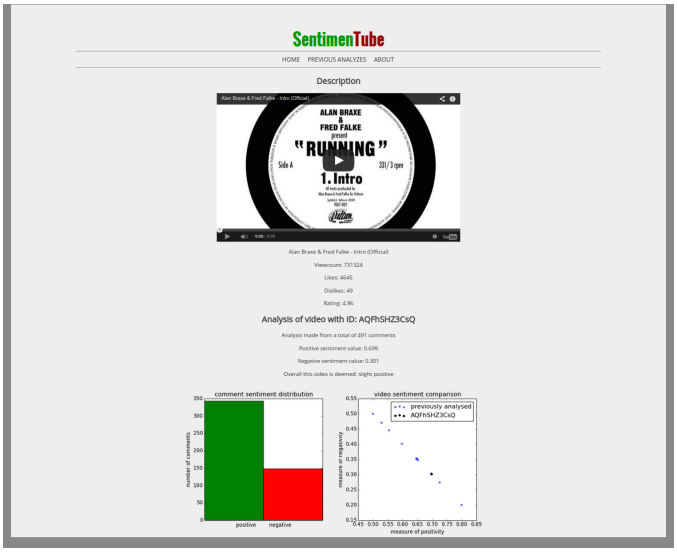

Figure 1. Result of sentiment analysis of YouTube video

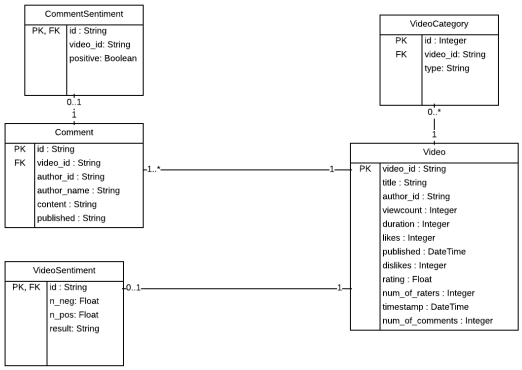

Figure 2. Database schema

Appendix A
Code listings

Listings


```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# pylint: disable=R0201
""" This module scrapes/download contents from a youtube video. """
import requests
import dateutil.parser
import logging
import datetime

import models


class YouTubeScraper:

    """ Class for communicating with the gdata youtube API. """

    def __init__(self):
        """ Set the gdata youtube urls and the logger. """
        self.comment_url = "https://gdata.youtube.com/feeds/api/videos/{0}/" \
                           "comments"
        self.video_url = "https://gdata.youtube.com/feeds/api/videos/{0}"
        self.logger = logging.getLogger(__name__)

    def _comment_generator(self, video_id):
        """
        A generator for fetching one "page" of youtube comments.

        For a youtube video, it returns a list of comment dictionaries
        with keys: author_name, author_id, content, video_id, id, published
        It should not be used directly, it is private and the fetch_comments
        method should be used instead.

        Parameters:
        - video_id: the id of the youtube video
        """
        next_url = self.comment_url.format(video_id)
        params = {"v": 2, "alt": "json", "max-results": 50,
                  "orderby": "published"}

        while True:
            if next_url:
                try:
                    response = requests.get(next_url, params=params)
                except requests.exceptions.RequestException:
                    self.logger.exception("_comment_generator: request failed")
                    raise
                else:
                    if not response:
                        error = "_comment_generator: invalid video id: " \
                            "{}".format(video_id)
                        self.logger.error(error)
                        raise ValueError(error)
                    response = response.json()
            else:
                raise StopIteration
            comments = []
            if "entry" not in response["feed"]:
                raise RuntimeError("no comments for video")
            for comment in response["feed"]["entry"]:
                author_name = comment["author"][0]["name"]["$t"]
                author_id = comment["author"][0]["yt$userId"]["$t"]
                content = comment["content"]["$t"]
                comment_id = comment["id"]["$t"]
                published = dateutil.parser.parse(comment["published"]["$t"])
```

```
65
66                  comment = models.Comment(id=comment_id,
67                                           video_id=video_id,
68                                           author_id=author_id,
69                                           author_name=author_name,
70                                           content=content,
71                                           published=published)
72
73                  comments.append(comment)
74              next_url = [link["href"] for link in response["feed"]["link"]
75                          if link["rel"] == "next"]
76              if next_url:
77                  next_url = next_url[0]
78              yield comments
79
80      def fetch_comments(self, video_id, number=0):
81          """
82  ␣␣␣␣␣␣␣␣fetch␣a␣number␣of␣youtube␣comments␣using␣_comment_generator.
83
84  ␣␣␣␣␣␣␣␣Parameters:
85  ␣␣␣␣␣␣␣␣–␣video_id␣:␣the␣id␣of␣the␣youtube␣video
86  ␣␣␣␣␣␣␣␣–␣number␣:␣the␣number␣of␣comments␣to␣fetch␣(0␣=␣all␣comments)
87
88  ␣␣␣␣␣␣␣␣Returns:
89  ␣␣␣␣␣␣␣␣–␣list␣of␣Comment␣objects
90  ␣␣␣␣␣␣␣␣"""
91          comments = []
92          fetch = self._comment_generator(video_id)
93          while True:
94              try:
95                  comments += next(fetch)
96                  if len(comments) > number and number > 0:
97                      return comments[:number]
98              except StopIteration:
99                  return comments
100
101     def fetch_videoinfo(self, video_id):
102         """
103 ␣␣␣␣␣␣␣␣fetch␣relevant␣information␣about␣the␣video␣from␣the␣gdata␣youtube␣API.
104
105 ␣␣␣␣␣␣␣␣Parameters:
106 ␣␣␣␣␣␣␣␣–␣video_id␣:␣the␣id␣of␣the␣youtube␣video
107
108 ␣␣␣␣␣␣␣␣Returns:
109 ␣␣␣␣␣␣␣␣–␣tuple␣of␣Video␣object␣and␣list␣of␣Category␣objects
110 ␣␣␣␣␣␣␣␣"""
111         req = requests.get(self.video_url.format(video_id),
112                            params={"v": 2, "alt": "json"})
113         if not req:
114             self.logger.error("fetch_videoinfo:␣invalid␣video␣id")
115             raise ValueError("invalid␣video␣id")
116
117         req = req.json()
118         comment_permission = [entry["permission"] for entry in
119                               req["entry"]["yt$accessControl"] if
120                               entry["action"] == "comment"][0]
121
122         if comment_permission == "denied":
123             self.logger.error("fetch_videoinfo:␣comments␣disallowed␣for␣video")
124
125             raise RuntimeError("Comments␣disallowed␣for␣video␣{0}".format(
126                 video_id))
127
128         video = self.extract_video(req, video_id)
129         categories = self.extract_categories(req, video_id)
130         return video, categories
131
132     def extract_categories(self, req, video_id):
133         """
134 ␣␣␣␣␣␣␣␣extract␣categories␣from␣a␣json–converted␣gdata␣video␣HTTP␣response.
135
136 ␣␣␣␣␣␣␣␣Parameters:
137 ␣␣␣␣␣␣␣␣–␣req␣:␣the␣gdata␣video␣HTTP␣response
138 ␣␣␣␣␣␣␣␣–␣video_id:␣the␣youtube␣video␣id
139 ␣␣␣␣␣␣␣␣Returns:
140 ␣␣␣␣␣␣␣␣–␣list␣of␣Category␣objects
141 ␣␣␣␣␣␣␣␣"""
142         categories = req["entry"]["media$group"]["media$category"]
143         categories = [models.VideoCategory(type=category["$t"],
144                                            video_id=video_id)
145                       for category in categories]
146         return categories
147
```

```
148         def extract_video(self, req, video_id):
149             """
150         extract video object from a json-converted gdata video HTTP response.
151 
152         Parameters:
153         - req: the gdata video HTTP response
154         - video_id: the youtube video id
155         Returns:
156         - a Video object
157         """
158             rating, numraters = None, None
159             if "gd$rating" in req["entry"]:
160                 rating = req["entry"]["gd$rating"]["average"]
161                 numraters = req["entry"]["gd$rating"]["numRaters"]
162             likes, dislikes = None, None
163             if "yt$rating" in req["entry"]:
164                 likes = req["entry"]["yt$rating"]["numLikes"]
165                 dislikes = req["entry"]["yt$rating"]["numDislikes"]
166 
167             title = req["entry"]["title"]["$t"]
168             author_id = req["entry"]["author"][0]["yt$userId"]["$t"]
169             viewcount = req["entry"]["yt$statistics"]["viewCount"]
170             duration = req["entry"]["media$group"]["media$content"][0]["duration"]
171             published = dateutil.parser.parse(req["entry"]["published"]["$t"])
172             num_of_comments = \
173                 req["entry"]["gd$comments"]["gd$feedLink"]["countHint"]
174             video = models.Video(id=video_id,
175                                  title=title,
176                                  author_id=author_id,
177                                  viewcount=viewcount,
178                                  duration=duration,
179                                  published=published,
180                                  rating=rating,
181                                  num_of_raters=numraters,
182                                  likes=likes,
183                                  dislikes=dislikes,
184                                  num_of_comments=num_of_comments,
185                                  timestamp=datetime.datetime.now())
186             return video
187 
188 if __name__ == '__main__':
189     YOU = YouTubeScraper()
190     print(YOU.fetch_comments("wrong_url"))
```

```
1   #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3   # pylint: disable=R0201
4   """
5   Module for sentiment analysis.
6 
7   This module has 3 purposes:
8   1: Can load an existing classifier from a pickle file
9   2: Train and save a classifier to a pickle file
10  3: Can classify multiple comments objects (from a list) and deduct an overall
11      classification of the video
12  The comments object, is the comments from the youtube video which want to be
13  classified.
14  """
15 
16  import nltk.classify.util
17  import pickle
18  import os
19  import logging
20  import models
21  from nltk.corpus import stopwords
22  CUSTOM_STOP_WORDS = ['band', 'they', 'them']
23 
24 
25  def create_word_list(text_words_tuples):
26      """
27      Create a big set with ALL of the words from the corpus.
28 
29      :param text_words_tuples: Tuple with all text and their sentiments
30      :return words_list: The big set with all the words
31      """
32      words_list = set()
33      for (words, _) in text_words_tuples:
34          for word in words:
35              words_list.add(word.lower())
36      return words_list
37 
38 
39  def create_tagged_text(tuples):
```

```
    """
    Create a list of tuples containing words of the text and its sentiment.

    :param tuples: Tuples with text (as strings) and its sentiment
    :return tuples_text: The list of tuples
    """
    tuple_text = []
    stop = stopwords.words('english')
    for (text, sentiment) in tuples:
        words = text.split()
        clean_word = ([i.lower() for i in words
                       if not i.lower() in stop])
        tuple_text.append((clean_word, sentiment))
    return tuple_text


class SentimentAnalysis:

    """ Class for making sentiment analysis of video comments. """

    def __init__(self, file_name):
        """ Call the load method to load the classifier from file. """
        corpus_path = "data/corpus.txt"
        self.logger = logging.getLogger(__name__)
        _, self.word_list = self.create_words_and_tuples(corpus_path)
        self.file_path = os.path.join(os.path.dirname(__file__), file_name)
        self.classifier = self.load_classifier(corpus_path)

    def load_corpus(self, file_name, split=","):
        """
        Load corpus from file and stores it in a tuple.

        :param file_name: Name of the corpus file
        :param split: How to split a line in the corpus (text vs. sentiment).
                      Default split: ','
        :return: pt and nt: Respectively positive and negative tuple
                            (text, sentiment)
        """
        self.logger.debug("Loading corpus file")
        file_path = os.path.join(os.path.dirname(__file__), file_name)
        try:
            with open(file_path, 'r') as read_file:
                lines = read_file.readlines()[1:]
        except(FileExistsError, LookupError):
            self.logger.error("I/O error: corpus file not found")
            raise
        else:
            sentiment_dict = {0: "negative", 1: "positive"}
            return [(line.split(split)[1].strip(),
                     sentiment_dict[int(line.split(split)[0])])
                    for line in lines]

    def _word_feats_extractor(self, doc):
        """
        Extract features from corpus.

        :param words: List of words from corpus
        :return: Dict of the words as keys and True/False as values
        """
        doc_words = set(doc)
        return dict([("contains({})".format(i), i in doc_words)
                     for i in self.word_list])

    def create_words_and_tuples(self, corpus_filename):
        """
        load corpus and create tagged text and word_list.

        tagged_text is the words of the text and their sentiment,
        word_list is the words in the corpus

        :param corpus_filename: the filepath of the corpus
        """
        text = self.load_corpus(corpus_filename, ";")
        tagged_text = create_tagged_text(text)
        word_list = create_word_list(tagged_text)
        return tagged_text, word_list

    def _train(self, corpus_filename):
        """
        Train the classifier.

        Training the Naïve Bayes classifier, by calling the following methods:
        - load_corpus
```

```
123 ␣␣␣␣␣␣␣␣-␣create_tagged_text
124 ␣␣␣␣␣␣␣␣-␣create_word_list
125 ␣␣␣␣␣␣␣␣-␣word_feats_extractor
126 ␣␣␣␣␣␣␣␣"""
127         tagged_text, _ = self.create_words_and_tuples(corpus_filename)
128 
129         self.logger.debug("Making␣training␣set␣(apply␣features)...")
130 
131         training_set = nltk.classify.apply_features(
132             self._word_feats_extractor, tagged_text)
133 
134         classifier = nltk.NaiveBayesClassifier.train(training_set)
135         self._save_classifier(classifier)
136         return classifier
137 
138     def _save_classifier(self, classifier):
139         """
140 ␣␣␣␣␣␣␣␣Save␣the␣classifier␣to␣a␣pickle␣file.
141 
142 ␣␣␣␣␣␣␣␣:param␣classifier:␣The␣trained␣classifier
143 ␣␣␣␣␣␣␣␣"""
144         try:
145             file_open = open(self.file_path, 'wb')
146             pickle.dump(classifier, file_open, 1)
147             file_open.close()
148             self.logger.info("Classifier␣saved␣successfully!")
149         except IOError:
150             self.logger.debug("Couldn't␣save␣the␣classifier␣to␣pickle")
151 
152     def load_classifier(self, corpus_path):
153         """
154 ␣␣␣␣␣␣␣␣Load␣a␣trained␣classifier␣from␣file.
155 
156 ␣␣␣␣␣␣␣␣If␣it␣fails,␣it's␣training␣a␣new
157 ␣␣␣␣␣␣␣␣"""
158         try:
159             classifier = nltk.data.load(
160                 "file:"+self.file_path, 'pickle', 1)
161             self.logger.info("Classifier␣loaded!")
162             return classifier
163         except (FileExistsError, LookupError):
164             self.logger.error("I/O␣error:␣classifier␣file␣not␣found")
165             self.logger.info("Will␣train␣a␣classifier")
166             return self._train(corpus_path)
167 
168     def classify_comments(self, comments):
169         """
170 ␣␣␣␣␣␣␣␣Classify␣youtube-videos␣comments.
171 
172 ␣␣␣␣␣␣␣␣performs␣classification␣on␣each␣comment
173 ␣␣␣␣␣␣␣␣and␣let␣the␣method␣'eval'␣make␣a␣decision
174 ␣␣␣␣␣␣␣␣It␣normalize␣the␣ratio␣between␣number␣of␣positive␣and␣negative␣comments
175 ␣␣␣␣␣␣␣␣before␣calling␣the␣'eval'␣method
176 ␣␣␣␣␣␣␣␣:param␣comments:␣The␣comments␣of␣youtube-video
177 ␣␣␣␣␣␣␣␣:return:
178 ␣␣␣␣␣␣␣␣"""
179         video_sentiment = models.VideoSentiment(id=comments[0].video_id,
180                                                 n_pos=0, n_neg=0, result="")
181 
182         self.logger.info(
183             "There␣is␣a␣change␣in␣comments.␣We␣do␣sentiment␣analysis")
184         comments_sentiment = []
185         for comment in comments:
186             res = self.classifier.classify(self._word_feats_extractor(
187                 comment.content.split()))
188 
189             if res == "pos":
190                 video_sentiment.n_pos += 1
191                 comments_sentiment.append(models.CommentSentiment(
192                     id=comment.id, video_id=comment.video_id, positive=1))
193             else:
194                 video_sentiment.n_neg += 1
195                 comments_sentiment.append(models.CommentSentiment(
196                     id=comment.id, video_id=comment.video_id, positive=0))
197 
198         total_data = video_sentiment.n_pos + video_sentiment.n_neg
199         self.logger.debug("Number␣of␣negative␣comments:␣%d",
200                           video_sentiment.n_neg)
201         self.logger.debug("Number␣of␣positive␣comments:␣%d",
202                           video_sentiment.n_pos)
203 
204         video_sentiment.n_pos /= total_data
205         video_sentiment.n_neg /= total_data
```

```
206             self.logger.debug("Number of negative comments after "
207                               "normalization: %.2f", video_sentiment.n_neg)
208             self.logger.debug(
209                 "Number of positive comments after normalization: %.2f",
210                 video_sentiment.n_pos)
211
212             video_sentiment.result = self._eval(video_sentiment)
213             self.logger.info("The result of the video: %s",
214                              video_sentiment.result)
215             return video_sentiment, comments_sentiment
216
217         def _eval(self, video_sentiment):
218             """
219         evaluate the overall sentiment of the youtube video.
220
221         Taking a decision of the sentiment of the youtube-video
222         based on the ratio between positive and negative comments
223         It takes a decision like so, based on number positive comments (nPos):
224         ------------ nPos < 0.25:     Strong negative
225         ------------ nPos >= 0.25 and nPos < 0.4:    Slight negative
226         ------------ nPos >= 0.4 and nPos < 0.6:     Neutral
227         ------------ nPos >= 0.6 ann nPos < 0.75:    Slight positive
228         ------------ nPos >= 0.75:    Strong positive
229         :param video_sentiment:
230         :return:
231         """
232             if video_sentiment.n_pos < .25:
233                 res = "strong negative"
234             elif video_sentiment.n_pos >= .25 and video_sentiment.n_pos < .4:
235                 res = "slight negative"
236             elif video_sentiment.n_pos >= .4 and video_sentiment.n_pos < .6:
237                 res = "neutral"
238             elif video_sentiment.n_pos >= .6 and video_sentiment.n_pos < .75:
239                 res = "slight positive"
240             else:
241                 res = "strong positive"
242
243             return res
```

```
1   #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3   """
4   Flask app for webservice.
5
6   handles the interacting between the user and the system.
7   """
8   import flask
9   from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
10  from matplotlib.figure import Figure
11  import io
12  import urllib
13  import urllib.parse
14  import logging
15  import sqlalchemy
16
17  import database
18  import models
19  import sentiment_analysis
20  import youtube
21
22  logging.basicConfig(format="%(asctime)s %(message)s", level=logging.DEBUG)
23  LOGGER = logging.getLogger(__name__)
24
25  ANALYZER = sentiment_analysis.SentimentAnalysis("data/classifier.pickle")
26  SCRAPER = youtube.YouTubeScraper()
27
28  APP = flask.Flask(__name__)
29
30
31  def save_sentiment(video_sentiment, comments_sentiment):
32      """
33      helper function for saving sentiments in the database.
34
35      Saves the results of sentiment analysis to the database.
36      The result of each comment and for the whole video is saved
37      :param video_sentiment: sentiment result for the whole video:
38      number of pos and neg comments (normalized) and final verdict of the video
39      :param comments_sentiment: comments of the video with their sentiments
40      """
41      db_sentiments = database.DB_SESSION.query(models.CommentSentiment).filter(
42          models.CommentSentiment.video_id == video_sentiment.id).all()
43      db_comment_sentiment_ids = [db_comment.id for db_comment in db_sentiments]
44
```

```
    for comment_sentiment in comments_sentiment:
        if comment_sentiment.id not in db_comment_sentiment_ids:
            database.DB_SESSION.add(comment_sentiment)

    db_videosentiment = database.DB_SESSION.query(
        models.VideoSentiment).filter(
            models.VideoSentiment.id == video_sentiment.id).first()

    if db_videosentiment:
        db_videosentiment = database.DB_SESSION.merge(video_sentiment)
    else:
        database.DB_SESSION.add(video_sentiment)
    database.DB_SESSION.commit()


@APP.route("/")
def index():
    """
    Show the front page to the user.

    :return: the front page (index.html)
    """
    return flask.render_template("index.html")


@APP.route("/about")
def about():
    """
    Show the about page to the user.

    :return: the about page (about.html)
    """
    return flask.render_template("about.html")


@APP.route("/video")
def video():
    """
    Video analysis page.

    Run the classification for the input the user has given
    Checks in database whether the video has been processed before. If it has
    been processed before and there is no changes, it simply shows the result.
    Else, it will process the video and show the result
    :return: The video page (video.html) with the result from database or
             classification.
    """
    video_id = flask.request.args.get("video_id")
    # if in the form of an url, extract id
    if "youtube" in video_id:
        url = urllib.parse.urlparse(video_id)
        query = dict(urllib.parse.parse_qsl(url[4]))
        video_id = query["v"]

    db_video_info = database.DB_SESSION.query(models.Video).filter(
        models.Video.id == video_id).first()
    try:
        video_info, categories = SCRAPER.fetch_videoinfo(video_id)
    except ValueError:
        return flask.render_template("error.html",
                                     error="invalid video id")
    except RuntimeError as err:
        return flask.render_template("error.html", error=str(err))

    if (db_video_info and
            db_video_info.num_of_comments == video_info.num_of_comments):
        LOGGER.info("sentiment for video with id: %r found in database",
                    video_id)

        sentiment = database.DB_SESSION.query(models.VideoSentiment).filter(
            models.VideoSentiment.id == video_id).first()

        comments = database.DB_SESSION.query(models.Comment).filter(
            models.Comment.video_id == video_id).all()
    else:
        LOGGER.info("processing new video with id: %r", video_id)
        if db_video_info:
            db_video_info = database.DB_SESSION.merge(video_info)
        else:
            database.DB_SESSION.add(video_info)
            database.DB_SESSION.add_all(categories)
        try:
            comments = SCRAPER.fetch_comments(video_id)
```

```
128             except RuntimeError as err:
129                 return flask.render_template("error.html", error=str(err))
130 
131             # get unique comments only
132             unique_ids = set([comment.id for comment in comments])
133             comments = [next(com for com in comments if com.id == com_id)
134                         for com_id in unique_ids]
135 
136             for comment in comments:
137                 comment_in_db = database.DB_SESSION.query(models.Comment).filter(
138                     models.Comment.id == comment.id).first()
139 
140                 if not comment_in_db:
141                     database.DB_SESSION.add(comment)
142 
143             database.DB_SESSION.commit()
144 
145             sentiment, comment_sentiments = ANALYZER.classify_comments(comments)
146 
147             save_sentiment(sentiment, comment_sentiments)
148         video_dict = {"sentiment": sentiment, "video_info": video_info,
149                       "num_of_comments": len(comments)}
150         return flask.render_template("video.html", video=video_dict)
151 
152 
153 @APP.errorhandler(404)
154 def not_found(error):
155     """
156     Show an error message to the user.
157 
158     :param error:
159     :return: The error page with the message
160     """
161     return flask.render_template("error.html", error=error)
162 
163 
164 @APP.route("/previous")
165 def previous():
166     """ return 5 latest sentiment analyses. """
167     latest = database.DB_SESSION.query(models.Video).order_by(
168         sqlalchemy.desc(models.Video.timestamp)).limit(5)
169 
170     return flask.render_template("previous.html", latest=latest)
171 
172 
173 @APP.route("/comment_sentiment_plot.png")
174 def comment_sentiment_plot():
175     """
176     Create comment sentiment plot.
177 
178     Creating the histogram-plot for the sentiments of the comments of the video
179     :return: PNG file showing the histogram
180     """
181     video_id = flask.request.args.get("video_id")
182     fig = Figure(figsize=(5, 5))
183     axis = fig.add_subplot(1, 1, 1)
184     fig.patch.set_alpha(0)
185 
186     query = database.DB_SESSION.query(models.CommentSentiment).filter(
187         models.CommentSentiment.video_id == video_id).all()
188     positive = [q.positive for q in query if q.positive]
189     negative = [q.positive for q in query if not q.positive]
190 
191     if positive:
192         axis.hist(positive, color=["g"], align="left", bins=[0, 1])
193     if negative:
194         axis.hist(negative, color=["r"], align="right", bins=[0, 1])
195     axis.set_xlabel("positive        negative")
196     axis.set_xticks([])
197     axis.set_title("comment sentiment distribution")
198     axis.set_ylabel("number of comments")
199     canvas = FigureCanvas(fig)
200     output = io.BytesIO()
201     canvas.print_png(output)
202     response = flask.make_response(output.getvalue())
203     response.mimetype = 'image/png'
204     return response
205 
206 
207 @APP.route("/video_sentiment_plot.png")
208 def video_sentiment_plot():
209     """
210     Create video sentiment plot.
```

```
    Creating a scatter-plot for the sentiments of the video against other
    videos with the same youtube-category
    :return: PNG file showing the scatter-plot
    """
    video_id = flask.request.args.get("video_id")
    fig = Figure(figsize=(5, 5))
    axis = fig.add_subplot(1, 1, 1)
    fig.patch.set_alpha(0)

    videos = database.DB_SESSION.query(models.VideoSentiment).all()

    current_video = database.DB_SESSION.query(models.VideoSentiment).filter(
        models.VideoSentiment.id == video_id).first()

    axis.scatter([v.n_pos for v in videos], [v.n_neg for v in videos],
                 color="blue", marker="x", label="previously analysed")
    axis.scatter(current_video.n_pos, current_video.n_neg, color="black",
                 marker="o", label=video_id)
    axis.set_title("video sentiment comparison")
    axis.set_xlabel("measure of positivity")
    axis.set_ylabel("measure of negativity")
    axis.legend()
    canvas = FigureCanvas(fig)
    output = io.BytesIO()
    canvas.print_png(output)
    response = flask.make_response(output.getvalue())
    response.mimetype = 'image/png'
    return response

if __name__ == "__main__":
    APP.run(debug=True)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# pylint: disable=W0232, R0903

""" database models for sqlalchemy. """
import sqlalchemy
from database import BASE


class Comment(BASE):

    """ Comment object. """

    __tablename__ = "comments"
    __table_args__ = {'extend_existing': True}
    id = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    video_id = sqlalchemy.Column(sqlalchemy.String,
                                 sqlalchemy.ForeignKey("videos.id"))
    author_id = sqlalchemy.Column(sqlalchemy.String, nullable=False)
    author_name = sqlalchemy.Column(sqlalchemy.String, nullable=False)
    content = sqlalchemy.Column(sqlalchemy.String, nullable=False)
    published = sqlalchemy.Column(sqlalchemy.DateTime, nullable=False)

    def __repr__(self):
        """ __repr__ method for Comment with necessary information. """
        return "{}(id={}, video_id={}, author_id={}, content={})".format(
            self.__class__.__name__, self.id, self.video_id, self.author_id,
            self.content)


class Video(BASE):

    """ Video object. """

    __tablename__ = "videos"
    __table_args__ = {'extend_existing': True}
    id = sqlalchemy.Column(sqlalchemy.String, primary_key=True, nullable=False)
    title = sqlalchemy.Column(sqlalchemy.String, nullable=False)
    author_id = sqlalchemy.Column(sqlalchemy.String, nullable=False)
    viewcount = sqlalchemy.Column(sqlalchemy.Integer, nullable=False)
    duration = sqlalchemy.Column(sqlalchemy.Integer, nullable=False)
    likes = sqlalchemy.Column(sqlalchemy.Integer, nullable=True)
    published = sqlalchemy.Column(sqlalchemy.DateTime, nullable=False)
    dislikes = sqlalchemy.Column(sqlalchemy.Integer, nullable=True)
    rating = sqlalchemy.Column(sqlalchemy.Float, nullable=True)
    num_of_raters = sqlalchemy.Column(sqlalchemy.Integer, nullable=True)
    timestamp = sqlalchemy.Column(sqlalchemy.DateTime, nullable=False)
    num_of_comments = sqlalchemy.Column(sqlalchemy.Integer,
                                        nullable=False)
```

```
51
52      def __repr__(self):
53          """ __repr__ method for Video. """
54          return "{}(id={}, title={}, author_id={})".format(
55              self.__class__.__name__, self.id, self.title, self.author_id)
56
57
58  class VideoSentiment(BASE):
59
60      """ VideoSentiment object. """
61
62      __tablename__ = "videosentiments"
63      __table_args__ = {'extend_existing': True}
64      id = sqlalchemy.Column(sqlalchemy.String,
65                             sqlalchemy.ForeignKey("videos.id"),
66                             primary_key=True, nullable=False)
67      n_pos = sqlalchemy.Column(sqlalchemy.Float, nullable=False)
68      n_neg = sqlalchemy.Column(sqlalchemy.Float, nullable=False)
69      result = sqlalchemy.Column(sqlalchemy.String, nullable=False)
70
71      def __repr__(self):
72          """ __repr__ method for VideoSentiment. """
73          return "{}(id={}, n_pos={:.3}, n_neg={:.3}, result={})".format(
74              self.__class__.__name__, self.id, self.n_pos, self.n_neg,
75              self.result)
76
77
78  class CommentSentiment(BASE):
79
80      """ CommentSentiment object. """
81
82      __tablename__ = "commentsentiments"
83      __table_args__ = {'extend_existing': True}
84      id = sqlalchemy.Column(sqlalchemy.String,
85                             sqlalchemy.ForeignKey("comments.id"),
86                             primary_key=True, nullable=False)
87      video_id = sqlalchemy.Column(sqlalchemy.String,
88                                   sqlalchemy.ForeignKey("videos.id"),
89                                   nullable=False)
90      positive = sqlalchemy.Column(sqlalchemy.Boolean, nullable=False)
91
92      def __repr__(self):
93          """ __repr__ method for CommentSentiment. """
94          return "{}(id={}, video_id={}, positive={})".format(
95              self.__class__.__name__, self.id, self.video_id, self.positive)
96
97
98  class VideoCategory(BASE):
99
100     """ VideoCategory object. """
101
102     __tablename__ = "videocategories"
103     __table_args__ = {'extend_existing': True}
104     id = sqlalchemy.Column(sqlalchemy.Integer,
105                            primary_key=True, nullable=False)
106     video_id = sqlalchemy.Column(sqlalchemy.String,
107                                  sqlalchemy.ForeignKey("videos.id"),
108                                  nullable=False)
109     type = sqlalchemy.Column(sqlalchemy.String, nullable=False)
110
111     def __repr__(self):
112         """ __repr__ method for VideoCategory. """
113         return "{}(id={}, video_id={}, type={})".format(
114             self.__class__.__name__, self.id, self.video_id, self.type)
```

```
1   #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3   """ Handling the database connection. """
4   import sqlalchemy
5   from sqlalchemy.ext.declarative import declarative_base
6   import os
7
8
9   CWDIR = os.path.join(os.path.dirname(__file__), "data", "project.db")
10  ENGINE = sqlalchemy.create_engine("sqlite:///{}".format(CWDIR), echo=False)
11  DB_SESSION = sqlalchemy.orm.scoped_session(sqlalchemy.orm.sessionmaker(
12      autocommit=False,
13      autoflush=False,
14      bind=ENGINE))
15  BASE = declarative_base()
16
17
18  def init_db():
```

```
    """ Create the database and its tables. """
    import models  # noqa # pylint: disable=unused-variable
    BASE.metadata.create_all(bind=ENGINE)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# pylint: disable=R0904

""" Module for integration testing the webserve module. """

from unittest import TestCase
import webserve
import database
import sqlalchemy
import models
import datetime


def insert_rows(video_ids=None, positive_list=None):
    """ Helper function for inserting test rows in the database. """
    if not video_ids:
        video_ids = ["tkXr3uxM2fy"]
    if not positive_list:
        positive_list = [True]
    for video_index, v_id in enumerate(video_ids):
        now = datetime.datetime.now()
        database.DB_SESSION.add(models.Video(id=v_id,
                                             title="test title {}".
                                             format(v_id),
                                             author_id="test author {}"
                                             .format(v_id),
                                             viewcount=1,
                                             duration=10,
                                             likes=1,
                                             published=now,
                                             dislikes=1,
                                             rating=3,
                                             num_of_raters=5,
                                             num_of_comments=5,
                                             timestamp=now))
        database.DB_SESSION.add(models.VideoSentiment(id=v_id,
                                                      n_pos=5.2,
                                                      n_neg=10.2,
                                                      result="negative"))
        for com_index, pos in enumerate(positive_list):
            database.DB_SESSION.add(models.Comment(id="comment {} {}"
                                                   .format(video_index,
                                                           com_index),
                                                   video_id=v_id,
                                                   author_id="test author "
                                                             "id {}"
                                                   .format(v_id),
                                                   author_name="test "
                                                               "author "
                                                               "name {}"
                                                   .format(v_id),
                                                   content="test comment"
                                                           " text {}"
                                                   .format(v_id),
                                                   published=now))

            database.DB_SESSION.add(
                models.CommentSentiment(
                    id="comment {} {}".format(video_index,
                                              com_index),
                    video_id=v_id, positive=pos))
    database.DB_SESSION.commit()


class WebServeTestCase(TestCase):

    """ Class to test webserve module. """
    def setUp(self):
        """
        setUp method for all tests.
        set up method, running before
        each test, sets up an in-memory sqlite database
        for use as test database and
        sets flask up for testing
        """
        webserve.APP.config["TESTING"] = True
        self.app = webserve.APP.test_client()
        database.ENGINE = sqlalchemy.create_engine("sqlite://", echo=False)
```

```
80          database.DB_SESSION = \
81              sqlalchemy.orm.scoped_session(sqlalchemy.orm
82                                            .sessionmaker(
83                                                autocommit=False,
84                                                autoflush=False,
85                                                bind=database.ENGINE))
86          database.init_db()
87
88      def tearDown(self):
89          """
90  ␣␣␣␣␣␣␣␣tearDown␣method␣for␣all␣tests.
91  ␣␣␣␣␣␣␣␣tear␣down␣method,␣running␣after
92  ␣␣␣␣␣␣␣␣each␣test,␣closes␣the␣session
93  ␣␣␣␣␣␣␣␣"""
94          database.DB_SESSION.close()
95
96      def test_start_page_load_correct(self):
97          """
98  ␣␣␣␣␣␣␣␣Test␣that␣the␣start␣page␣is␣loading␣correctly.
99
100 ␣␣␣␣␣␣␣␣asserts␣on␣text␣in␣index␣page
101 ␣␣␣␣␣␣␣␣"""
102         response = self.app.get("/")
103         assert "Enter␣ID␣or␣URL" in response.data.decode("utf-8")
104
105     def test_video_page_load_correct_from_database(self):
106         """
107 ␣␣␣␣␣␣␣␣Test␣that␣video␣loads␣from␣database␣directly␣if␣found.
108
109 ␣␣␣␣␣␣␣␣asserts␣on␣text␣on␣video␣analysis␣page
110 ␣␣␣␣␣␣␣␣"""
111         v_id = "tkXr3uxM2fY"
112         insert_rows([v_id])
113         response = self.app.get("/video?video_id={}".format(v_id))
114         assert "Analysis␣of␣video␣with␣ID:␣{}".format(v_id) in \
115                response.data.decode("utf-8")
116
117     def test_video_page_load_error_wrong_id(self):
118         """
119 ␣␣␣␣␣␣␣␣Test␣that␣tries␣to␣input␣an␣invalid␣video␣id␣at␣the␣start␣page.
120
121 ␣␣␣␣␣␣␣␣asserts␣on␣error␣text␣in␣video␣analysis␣page
122 ␣␣␣␣␣␣␣␣"""
123         response = self.app.get("/video?video_id={}".format("wrong_id"))
124         assert "Error:␣invalid␣video␣id" in response.data.decode("utf-8")
125
126     def test_video_page_load_correct_full_youtubeurl(self):
127         """
128 ␣␣␣␣␣␣␣␣Test␣of␣video␣page␣with␣URL.
129
130 ␣␣␣␣␣␣␣␣test␣that␣loads␣video␣page␣when␣given␣a␣full␣youtube␣url␣like:
131 ␣␣␣␣␣␣␣␣"https://www.youtube.com/watch?v=tkXr3uxM2fY"
132 ␣␣␣␣␣␣␣␣asserts␣on␣text␣in␣video␣analysis␣page
133 ␣␣␣␣␣␣␣␣"""
134         v_id = "tkXr3uxM2fY"
135         response = self.app.get(
136             "/video?video_id=https://www.youtube.com/watch?v={}".format(v_id))
137         assert "Analysis␣of␣video␣with␣ID:␣{}".format(v_id) in\
138             response.data.decode("utf-8")
139
140     def test_video_page_load_correct_from_youtube(self):
141         """
142 ␣␣␣␣␣␣␣␣Test␣that␣fetches␣youtube␣information␣and␣loads␣video␣page.
143
144 ␣␣␣␣␣␣␣␣this␣can␣be␣considered␣the␣"normal␣use␣case"
145 ␣␣␣␣␣␣␣␣asserts␣on␣text␣in␣video␣analysis␣page
146 ␣␣␣␣␣␣␣␣"""
147         v_id = "tkXr3uxM2fY"
148         response = self.app.get("/video?video_id={}".format(v_id))
149         assert "Analysis␣of␣video␣with␣ID:␣{}".format(v_id) in \
150                response.data.decode("utf-8")
151
152     def test_video_page_saves_video_in_db(self):
153         """
154 ␣␣␣␣␣␣␣␣Test␣asserting␣video␣is␣saved␣in␣database␣after␣video␣page␣load.
155
156 ␣␣␣␣␣␣␣␣asserts␣on␣test␣database␣query
157 ␣␣␣␣␣␣␣␣"""
158         v_id = "tkXr3uxM2fY"
159         self.app.get("/video?video_id={}".format(v_id))
160         vid = database.DB_SESSION.query(models.Video).filter_by(
161             id=v_id).first()
162         assert vid
```

```
163
164         def test_video_page_updates_sentiment_in_db(self):
165             """
166         ␣␣␣␣␣␣␣␣Test␣for␣"outdated"␣video␣in␣database.
167
168         ␣␣␣␣␣␣␣␣testing␣if␣sentiment␣is␣updated␣in␣the␣database
169         ␣␣␣␣␣␣␣␣if␣a␣video␣previously␣saved␣in␣database␣is␣updated
170         ␣␣␣␣␣␣␣␣at␣youtube␣(contains␣new␣comments)
171         ␣␣␣␣␣␣␣␣asserts␣on␣test␣database␣query
172         ␣␣␣␣␣␣␣␣"""
173             v_id = "tkXr3uxM2fY"
174             now = datetime.datetime.now()
175             negative_score = 100
176             positive_score = 100
177             database.DB_SESSION.add(models.VideoSentiment(id=v_id,
178                                                           n_neg=negative_score,
179                                                           n_pos=positive_score,
180                                                           result="test␣positive"))
181
182             database.DB_SESSION.add(models.Video(id=v_id, title="test␣title",
183                                                  author_id="test␣author␣id",
184                                                  viewcount=1, duration=5, likes=1,
185                                                  published=now,
186                                                  dislikes=1, rating=5,
187                                                  num_of_raters=1,
188                                                  timestamp=now,
189                                                  num_of_comments=10))
190             database.DB_SESSION.add(models.Comment(id="comment␣{}".format(v_id),
191                                                    video_id=v_id,
192                                                    author_id="test␣author␣id",
193                                                    author_name="test␣author",
194                                                    content="test␣comment",
195                                                    published=now))
196             database.DB_SESSION.commit()
197
198             self.app.get("/video?video_id={}".format(v_id))
199
200             sentiment = database.DB_SESSION.query(
201                 models.VideoSentiment).filter_by(id=v_id).first()
202             assert sentiment.n_neg != negative_score
203             assert sentiment.n_pos != positive_score
204             assert sentiment.result != "test␣positive"
205
206         def test_video_page_saves_comment_in_db(self):
207             """
208         ␣␣␣␣␣␣␣␣Test␣asserting␣comments␣are␣saved␣after␣video␣page␣load.
209
210         ␣␣␣␣␣␣␣␣asserts␣on␣test␣database␣query
211         ␣␣␣␣␣␣␣␣"""
212             v_id = "tkXr3uxM2fY"
213             self.app.get("/video?video_id={}".format(v_id))
214             comment = database.DB_SESSION.query(
215                 models.Comment).filter_by(video_id=v_id).first()
216             assert comment
217
218         def test_video_page_saves_commentsentiment_in_db(self):
219             """
220         ␣␣␣␣␣␣␣␣Test␣asserting␣comment␣sentiments␣are␣saved␣after␣video␣page␣load.
221
222         ␣␣␣␣␣␣␣␣asserts␣on␣test␣database␣query
223         ␣␣␣␣␣␣␣␣"""
224             v_id = "tkXr3uxM2fY"
225             self.app.get("/video?video_id={}".format(v_id))
226             sentiment = database.DB_SESSION.query(
227                 models.CommentSentiment).filter_by(video_id=v_id).first()
228             assert sentiment
229
230         def test_video_page_saves_videosentiment_in_db(self):
231             """
232         ␣␣␣␣␣␣␣␣Test␣asserting␣a␣videosentiment␣is␣saved␣after␣video␣page␣load.
233
234         ␣␣␣␣␣␣␣␣asserts␣on␣test␣database␣query
235         ␣␣␣␣␣␣␣␣"""
236             v_id = "tkXr3uxM2fY"
237             self.app.get("/video?video_id={}".format(v_id))
238             sentiment = database.DB_SESSION.query(
239                 models.VideoSentiment).filter_by(id=v_id).first()
240             assert sentiment
241
242         def test_video_page_comment_sentiment_plot_only_negative(self):
243             """
244         ␣␣␣␣␣␣␣␣Test␣asserting␣the␣comment␣sentiment␣plot␣works␣(with␣negative).
245
```

```
246         tests with only negative comment sentiments
247         """
248         v_id = "tkXr3uxM2fY"
249         insert_rows([v_id], positive_list=[False])
250         response = self.app.get("/comment_sentiment_plot.png?video_id={}"
251                                 .format(v_id))
252         assert response.status_code == 200
253 
254     def test_video_page_comment_sentiment_plot_only_positive(self):
255         """
256         Test asserting the comment sentiment plot works (with positive).
257 
258         tests with only positive comment sentiments
259         """
260         v_id = "tkXr3uxM2fY"
261         insert_rows([v_id], positive_list=[True])
262         response = self.app.get("/comment_sentiment_plot.png?video_id={}"
263                                 .format(v_id))
264         assert response.status_code == 200
265 
266     def test_video_page_comment_sentiment_plot_mixed(self):
267         """
268         Test asserting the comment sentiment plot works (mixed).
269 
270         with mixed comment sentiments (positive and negative)
271         """
272         v_id = "tkXr3uxM2fY"
273         insert_rows([v_id], positive_list=[True, False])
274         response = self.app.get("/comment_sentiment_plot.png?video_id={}"
275                                 .format(v_id))
276         assert response.status_code == 200
277 
278     def test_video_page_video_sentiment_plot_correct(self):
279         """
280         Test that video sentiment on video page load works correctly.
281 
282         asserts on HTTP status = 200
283         """
284         v_id = "tkXr3uxM2fY"
285         insert_rows([v_id], [True, False])
286         response = self.app.get("/video_sentiment_plot.png?video_id={}"
287                                 .format(v_id))
288         assert response.status_code == 200
289 
290     def test_previous_page_taking_newest(self):
291         """
292         Test that the previous page shows the 5 most recent analyses.
293 
294         inserts 10 test result and asserts on 5 last ids
295         """
296         v_ids = ["5nO7IA1DeeI", "vykkfDITkQs",
297                  "C3zqYM3Rkpg", "0piaF7P3404",
298                  "Ek_cufWYvjE", "zNJJBD_I5EU",
299                  "v2zTVZFlCZ0", "ZLa6sX9N3Jw",
300                  "c6qOBFkvdG0", "eYhHyUU-CYU"]
301         insert_rows(v_ids)
302 
303         response = self.app.get("/previous")
304         for v_id in v_ids[5:]:
305             assert v_id in response.data.decode("utf-8")
306 
307     def test_about_page_load_correct(self):
308         """
309         Test about page loads correctly.
310 
311         asserts on text on about page
312         """
313         response = self.app.get("/about")
314         assert "Created by Sören Howe Gersager and Anders Rahbek" in \
315                response.data.decode("utf-8")
316 
317     def test_error_page_load_from_wrong_url(self):
318         """
319         Test that webservice fails gracefully on wrong url.
320 
321         ensures an appropriate response is returned
322         when trying to load a page that does not exist
323         """
324         response = self.app.get("/verywrongurl")
325         assert "Error: 404: Not Found" in \
326                response.data.decode("utf-8")
327 
328     def test_video_page_error_disallowed_comments_video(self):
```

```
        """
        Test edge-case: video with comments disallowed.

        Test that ensures an appropriate response is returned
        when trying to analyze a video with comments disabled
        """
        response = self.app.get("/video?video_id={}".format("NZQQdlPoz5g"))
        assert "Error: Comments disallowed for video" in \
               response.data.decode("utf-8")

    def test_video_page_error_no_comments_video(self):
        """
        Test edge-case: video with no comments.

        Test that ensures an appropriate response is returned
        when trying to analyze a video with no comments

        asserts on error text
        """
        response = self.app.get("/video?video_id={}".format("wv4ol_Q4G_k"))
        assert "Error: no comments for video" in \
               response.data.decode("utf-8")
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# pylint: disable=W0212, R0201

""" tests for the youtube module. """

from unittest import mock, TestCase
import youtube
import models
import requests


class YouTubeTestCase(TestCase):

    """ This class has test-methods for youtube module. """

    @mock.patch("youtube.YouTubeScraper._comment_generator")
    def test_fetch_comments_correct_id(self, mock_comment):
        """
        test of fetch_comments in YouTubeScraper with correct id.

        calling the the fetch_comments method in the YouTubeScraper with
        a correct id and asserting the results
        :param mock_comment: Mock object for comment method. The method is not
                             been called
        """
        scraper = youtube.YouTubeScraper()
        # return dummy list
        cm = models.Comment(id="test", video_id="test", author_id="test",
                            author_name="test", content="test",
                            published="test")
        mock_comment.return_value = iter([[cm for x in range(100)]])
        scraper.fetch_comments("dQw4w9WgXcQ", 50)
        scraper._comment_generator.assert_called_with("dQw4w9WgXcQ")

    @mock.patch("youtube.YouTubeScraper._comment_generator")
    def test_fetch_comments_returns_correct_over_zero(self, mock_comment):
        """
        test of fetch_comments returning all comments correctly.

        this method tests that the correct amount of comments is returned
        when specified
        :param mock: Mock object for _comment_generator
        """
        scraper = youtube.YouTubeScraper()
        # return dummy list
        cm = models.Comment(id="test", video_id="test", author_id="test",
                            author_name="test", content="test",
                            published="test")
        mock_comment.return_value = iter([[cm for x in range(500)]])
        comments = scraper.fetch_comments("dQw4w9WgXcQ", 250)
        assert len(comments) == 250

    @mock.patch("youtube.YouTubeScraper._comment_generator")
    def test_fetch_comments_returns_all_at_zero(self, mock_comment):
        """
        test fetch_comments returning all comments unexplicitly.

        tests that all comments are returned when a number
```

```
        is not explicitly specified
        :param mock_comment: Mock object for _comment_generator
        """
        scraper = youtube.YouTubeScraper()
        # return dummy list
        cm = models.Comment(id="test", video_id="test", author_id="test",
                            author_name="test", content="test",
                            published="test")
        mock_comment.return_value = iter([[cm for x in range(500)]])
        comments = scraper.fetch_comments("dQw4w9WgXcQ")
        assert len(comments) == 500

    @mock.patch("logging.Logger.error")
    def test_comment_generator_wrong_videoid_gracefully(self, mock_logger):
        """
        test _comment_generator handles wrong video_id.

        tests that _comment_generator raises a ValueError
        when supplied with an invalid video_id
        :param mock_logger: Mock object for logger
        """
        scraper = youtube.YouTubeScraper()
        generator = scraper._comment_generator("wrong url")
        self.assertRaises(ValueError, next, generator)
        mock_logger.assert_called()

    @mock.patch("logging.Logger.error")
    def test_fetch_videoinfo_wrong_videoid_gracefully(self, mock_logger):
        """
        test fetch_videoinfo handles wrong video_id.

        tests that fetch_videoinfo raises exception
        and error is logged when supplied with an invalid video_id
        :param mock_logger: Mock object for logger
        """
        scraper = youtube.YouTubeScraper()
        self.assertRaises(ValueError, scraper.fetch_videoinfo, "wrong url")
        mock_logger.assert_called()

    @mock.patch("logging.Logger.exception")
    @mock.patch("requests.get")
    def test_fetchcomments_no_connection(self, mock_requests, mock_logger):
        """
        test in case of requests error.

        tests that connection error (requests) is logged
        :param mock_requests : Mock object for requests.get method
        :param mock_logger : Mock object for logger
        """
        scraper = youtube.YouTubeScraper()
        mock_requests.side_effect = requests.exceptions.RequestException
        assert mock_logger.assert_called()
        self.assertRaises(requests.exceptions.RequestException,
                          scraper.fetch_comments, "dQw4w9WgXcQ")
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# pylint: disable=W0212, R0201

""" Tests for the module sentiment_analysis. """
from unittest import mock, TestCase
import sentiment_analysis
import models
from datetime import datetime


class SentimentAnalysisTestCase(TestCase):

    """ This class has test-methods for sentiment_analysis module. """

    @mock.patch("sentiment_analysis.SentimentAnalysis._train")
    @mock.patch("sentiment_analysis.SentimentAnalysis.load_classifier")
    @mock.patch("nltk.data.load")
    def test_load_classifier(self, train, load_classifier, load_data):
        """
        Test the load_classifier method.

        :param train:
        :param load_classifier:
        :param load_data:
        :return:
        """
```

```
29          sentiment_analysis.SentimentAnalysis("data/classifier.pickle")
30          load_classifier.assert_called()
31          load_data.return_value = True
32          train.assert_not_called()
33
34      def test_classify_comments(self):
35          """ Test the classify_comment method in sentiment_analysis. """
36          comments = []
37          static_comments = ["I love you!", "I hate you!",
38                             "I wont talk to you. Idiot!", "you are sweet",
39                             "I'm happy", "i'm mad!"]
40
41          for comment in static_comments:
42              comments.append(models.Comment(video_id="dQw4w9WgXcQ",
43                                             author_id="xxx", author_name="yyy",
44                                             content=comment,
45                                             published=datetime.now()))
46
47          sa = sentiment_analysis.SentimentAnalysis("data/classifier.pickle")
48          video_sentiment, comments_sentiment = sa.classify_comments(
49              comments)
50          assert [com.positive for com in comments_sentiment] == [1, 0, 0, 1, 1,
51                                                                  0]
52
53          assert video_sentiment.n_pos == 0.5
54          assert video_sentiment.n_neg == 0.5
55
56      def test_eval(self):
57          """ Test the eval method in sentiment_analysis. """
58          test_ratios_pos = [0.1, 0.25, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.85,
59                             0.96]
60          test_objects = []
61          for ratio in test_ratios_pos:
62              test_objects.append(models.VideoSentiment(id="dQw4w9WgXcQ",
63                                                        n_pos=ratio,
64                                                        n_neg=(1-ratio),
65                                                        result=""))
66
67          sa = sentiment_analysis.SentimentAnalysis("data/classifier.pickle")
68          result = []
69          for object_test in test_objects:
70              result.append(sa._eval(object_test))
71
72          assert [res for res in result] == ["strong negative",
73                                             "slight negative",
74                                             "slight negative",
75                                             "neutral",
76                                             "neutral",
77                                             "neutral",
78                                             "slight positive",
79                                             "slight positive",
80                                             "strong positive",
81                                             "strong positive"]
82
83      @mock.patch("nltk.data.load")
84      @mock.patch("sentiment_analysis.SentimentAnalysis._train")
85      @mock.patch("logging.Logger")
86      def test_load_wrong_file(self, nltk_load, train, logger):
87          """
88  ________Test load method.
89
90  ________tests with wrong file-name
91  ________(or the file doesn't exist)
92
93  ________:param nltk_load: Mock object on nltk.load method with side_effect
94  ________:param train: Mock object for train method. The method is not been
95  ____________________called
96  ________:param logger: Mock object on logging method
97  ________"""
98          nltk_load.side_effect = (FileExistsError, LookupError)
99
100         sentiment_analysis.SentimentAnalysis(
101             "data/hello_hello.pickle")
102
103         logger.assert_called()
104         self.assertRaises(FileExistsError, nltk_load)
105         train.assert_called()
```

```
1   #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   """
5   This module contains tests of code formats.
```

```
6
7   - Flake8
8   - Unittest
9   - Pylint
10
11  The Unittest is defined individually in modules pr. modules wanted to be
12  tested
13  """
14  import unittest
15  import os
16  import subprocess
17
18  CWD = os.path.dirname(__file__)
19
20
21  class TestCodeFormat(unittest.TestCase):
22
23      """    Creating, listing and running tests. """
24
25      def test_pylint_compliance(self):
26          """ Test the modules for pylint violations. """
27          cmd = ["pylint", "--rcfile={}".format(os.path.join(CWD, os.pardir,
28                                                             "pylint.rc")),
29                 "sentimentube", "test"]
30          try:
31              subprocess.check_output(cmd, universal_newlines=True)
32          except subprocess.CalledProcessError as error:
33              errors = [line for line in error.output.split("\n")
34                        if not line.startswith("*") and
35                        "Locally disabling" not in line]
36              if errors:
37                  for error in errors:
38                      print(error)
39                  self.assertTrue(False, msg="pylint fail")
40
41      def test_flake8_compliance(self):
42          """ Test the modules for flake8 violations. """
43          cmd = ["flake8", "sentimentube", "test"]
44          try:
45              subprocess.check_output(cmd, universal_newlines=True)
46          except subprocess.CalledProcessError as err:
47              errors = err.output.split("\n")
48              if errors:
49                  for error in errors:
50                      print(error)
51                  self.assertTrue(False, msg="flake8 fail")
52
53      def test_pep257_compliance(self):
54          """ Test the modules for pep257 violations. """
55          cmd = ["pep257", "sentimentube", "test"]
56          try:
57              subprocess.check_output(cmd, universal_newlines=True)
58          except subprocess.CalledProcessError as err:
59              errors = err.output.split("\n")
60              if errors:
61                  for error in errors:
62                      print(error)
63                  self.assertTrue(False, msg="pep257 fail")
```

APPENDIX B
AUTOMATIC GENERATION OF DOCUMENTATION

Test output (nosetests):

```
nose.config: INFO: Ignoring files matching [’^\\.’, ’^_’, ’^setup\\.py$’]
nose.plugins.cover: INFO: Coverage report will include only packages: [’youtube’, ’webserve’,
nose.selector: INFO: /home/syre/Dropbox/opgaver/Kandidat/02819 Data mining med Python/datamini
Test the modules for flake8 violations. ... ok
Test the modules for pep257 violations. ... ok
Test the modules for pylint violations. ... ok
Test about page loads correctly. ... ok
Test that webservice fails gracefully on wrong url. ... ok
Test that the previous page shows the 5 most recent analyses. ... ok
Test that the start page is loading correctly. ... ok
Test asserting the comment sentiment plot works (mixed). ... ok
Test asserting the comment sentiment plot works (with negative). ... ok
Test asserting the comment sentiment plot works (with positive). ... ok
Test edge-case: video with comments disallowed. ... ok
Test edge-case: video with no comments. ... ok
Test that video loads from database directly if found. ... ok
Test that fetches youtube information and loads video page. ... ok
Test of video page with URL. ... ok
Test that tries to input an invalid video id at the start page. ... ok
Test asserting comments are saved after video page load. ... ok
Test asserting comment sentiments are saved after video page load. ... ok
Test asserting video is saved in database after video page load. ... ok
Test asserting a videosentiment is saved after video page load. ... ok
Test for "outdated" video in database. ... ok
Test that video sentiment on video page load works correctly. ... ok
Test the classify_comment method in sentiment_analysis. ... ok
Test the eval method in sentiment_analysis. ... ok
Test the load_classifier method. ... ok
Test load method. ... ok
test _comment_generator handles wrong video_id. ... ok
test of fetch_comments in YouTubeScraper with correct id. ... ok
test fetch_comments returning all comments unexplicitly. ... ok
test of fetch_comments returning all comments correctly. ... ok
test fetch_videoinfo handles wrong video_id. ... ok
test in case of requests error. ... ok

Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
database                11      0   100%
sentiment_analysis     104      0   100%
webserve               116      0   100%
youtube                 86      0   100%
--------------------------------------------------
TOTAL                  317      0   100%
----------------------------------------------------------------------
Ran 32 tests in 93.541s

OK
```

Epydoc documentation:

sentimentube

API Documentation

December 2, 2014

Contents

1 Package sentimentube

Initfile for pylint.

1.1 Modules

- **database**: Handling the database connection.
  (Section 2, p. 3)
- **models**: database models for sqlalchemy.
  (Section 3, p. 4)
- **sentiment_analysis**: Module for sentiment analysis.
  (Section 4, p. 8)
- **webserve**: Flask app for webservice.
  (Section 5, p. 10)
- **youtube**: This module scrapes/download contents from a youtube video.
  (Section 6, p. 12)

1.2 Variables

Name	Description
___package___	**Value:** `None`

2 Module sentimentube.database

Handling the database connection.

2.1 Functions

init_db()

Create the database and its tables.

2.2 Variables

Name	Description
CWDIR	**Value:** `os.path.join(os.path.dirname(__file__), "data", "project....`
ENGINE	**Value:** `sqlalchemy.create_engine("sqlite:///{}".format(CWDIR), ec...`
DB_SESSION	**Value:** `sqlalchemy.orm.scoped_session(sqlalchemy.orm.sessionmaker...`
BASE	**Value:** `declarative_base()`

3 Module sentimentube.models

database models for sqlalchemy.

3.1 Class Comment

declarative_base()

sentimentube.models.Comment

Comment object.

3.1.1 Methods

___repr___(*self*)

___repr___ method for Comment with necessary information.

3.1.2 Class Variables

Name	Description
___tablename___	**Value:** `"comments"`
___table_args___	**Value:** `{'extend_existing': True}`
id	**Value:** `sqlalchemy.Column(sqlalchemy.String, primary_key= True)`
video_id	**Value:** `sqlalchemy.Column(sqlalchemy.String, sqlalchemy.ForeignKe...`
author_id	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`
author_name	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`
content	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`
published	**Value:** `sqlalchemy.Column(sqlalchemy.DateTime, nullable= False)`

3.2 Class Video

declarative_base()

sentimentube.models.Video

Video object.

3.2.1 Methods

___repr___(*self*)
___repr___ method for Video.

3.2.2 Class Variables

Name	Description
___tablename___	**Value:** `"videos"`
___table_args___	**Value:** `{'extend_existing': True}`
id	**Value:** `sqlalchemy.Column(sqlalchemy.String, primary_key= True, n...`
title	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`
author_id	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`
viewcount	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= False)`
duration	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= False)`
likes	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= True)`
published	**Value:** `sqlalchemy.Column(sqlalchemy.DateTime, nullable= False)`
dislikes	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= True)`
rating	**Value:** `sqlalchemy.Column(sqlalchemy.Float, nullable= True)`
num_of_raters	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= True)`
timestamp	**Value:** `sqlalchemy.Column(sqlalchemy.DateTime, nullable= False)`
num_of_comments	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, nullable= False)`

3.3 Class VideoSentiment

declarative_base() ┐
sentimentube.models.VideoSentiment

VideoSentiment object.

3.3.1 Methods

___repr___(*self*)
___repr___ method for VideoSentiment.

3.3.2 Class Variables

Name	Description
___tablename___	**Value:** `"videosentiments"`
___table_args___	**Value:** `{'extend_existing': True}`
id	**Value:** `sqlalchemy.Column(sqlalchemy.String, sqlalchemy.ForeignKe...`
n_pos	**Value:** `sqlalchemy.Column(sqlalchemy.Float, nullable= False)`
n_neg	**Value:** `sqlalchemy.Column(sqlalchemy.Float, nullable= False)`
result	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`

3.4 Class CommentSentiment

declarative_base() ┐
sentimentube.models.CommentSentiment

CommentSentiment object.

3.4.1 Methods

___repr___(*self*)
___repr___ method for CommentSentiment.

3.4.2 Class Variables

Name	Description
___tablename___	**Value:** `"commentsentiments"`
___table_args___	**Value:** `{'extend_existing': True}`
id	**Value:** `sqlalchemy.Column(sqlalchemy.String, sqlalchemy.ForeignKe...`
video_id	**Value:** `sqlalchemy.Column(sqlalchemy.String, sqlalchemy.ForeignKe...`
positive	**Value:** `sqlalchemy.Column(sqlalchemy.Boolean, nullable= False)`

3.5 Class VideoCategory

declarative_base()
sentimentube.models.VideoCategory

VideoCategory object.

3.5.1 Methods

___repr___(*self*)
___repr___ method for VideoCategory.

3.5.2 Class Variables

Name	Description
___tablename___	**Value:** `"videocategories"`
___table_args___	**Value:** `{'extend_existing': True}`
id	**Value:** `sqlalchemy.Column(sqlalchemy.Integer, primary_key= True, ...`
video_id	**Value:** `sqlalchemy.Column(sqlalchemy.String, sqlalchemy.ForeignKe...`
type	**Value:** `sqlalchemy.Column(sqlalchemy.String, nullable= False)`

4 Module sentimentube.sentiment_analysis

```
Module for sentiment analysis.

This module has 3 purposes:
1: Can load an existing classifier from a pickle file
2: Train and save a classifier to a pickle file
3: Can classify multiple comments objects (from a list) and deduct an overall
   classification of the video
The comments object, is the comments from the youtube video which want to be
classified.
```

4.1 Functions

create_word_list(*text_words_tuples*)

Create a big set with ALL of the words from the corpus.

:param text_words_tuples: Tuple with all text and their sentiments :return words_list: The big set with all the words

create_tagged_text(*tuples*)

Create a list of tuples containing words of the text and its sentiment.

:param tuples: Tuples with text (as strings) and its sentiment :return tuples_text: The list of tuples

4.2 Variables

Name	Description
CUSTOM_STOP_WORDS	**Value:** `['band', 'they', 'them']`

4.3 Class SentimentAnalysis

Class for making sentiment analysis of video comments.

4.3.1 Methods

___init___(*self*, *file_name*)

Call the load method to load the classifier from file.

load_corpus(*self*, *file_name*, *split*=",")

```
Load corpus from file and stores it in a tuple.

:param file_name: Name of the corpus file
:param split: How to split a line in the corpus (text vs. sentiment).
              Default split: ','
:return: pt and nt: Respectively positive and negative tuple
                    (text, sentiment)
```

create_words_and_tuples(*self*, *corpus_filename*)

load corpus and create tagged text and word_list.

tagged text is the words of the text and their sentiment, word_list is the words in the corpus

:param corpus_filename: the filepath of the corpus

load_classifier(*self*, *corpus_path*)

Load a trained classifier from file.

If it fails, it's training a new

classify_comments(*self*, *comments*)

Classify youtube-videos comments.

performs classification on each comment and let the method 'eval' make a decision It normalize the ratio between number of positive and negative comments before calling the 'eval' method :param comments: The comments of youtube-video :return:

5 Module sentimentube.webserve

Flask app for webservice.

handles the interacting between the user and the system.

5.1 Functions

save_sentiment(*video_sentiment*, *comments_sentiment*)

helper function for saving sentiments in the database.

Saves the results of sentiment analysis to the database. The result of each comment and for the whole video is saved :param video_sentiment: sentiment result for the whole video: number of pos and neg comments (normalized) and final verdict of the video :param comments_sentiment: comments of the video with their sentiments

index()

Show the front page to the user.

:return: the front page (index.html)

about()

Show the about page to the user.

:return: the about page (about.html)

video()

```
Video analysis page.

Run the classification for the input the user has given
Checks in database whether the video has been processed before. If it has
been processed before and there is no changes, it simply shows the result.
Else, it will process the video and show the result
:return: The video page (video.html) with the result from database or
        classification.
```

not_found(*error*)

Show an error message to the user.

:param error: :return: The error page with the message

previous()

return 5 latest sentiment analyses.

comment_sentiment_plot()

Create comment sentiment plot.

Creating the histogram-plot for the sentiments of the comments of the video :return: PNG file showing the histogram

video_sentiment_plot()

Create video sentiment plot.

Creating a scatter-plot for the sentiments of the video against other videos with the same youtube-category :return: PNG file showing the scatter-plot

5.2 Variables

Name	Description
LOGGER	**Value:** `logging.getLogger(__name__)`
ANALYZER	**Value:** `sentiment_analysis.SentimentAnalysis("data/classifier.pic...`
SCRAPER	**Value:** `youtube.YouTubeScraper()`
APP	**Value:** `flask.Flask(__name__)`

6 Module sentimentube.youtube

This module scrapes/download contents from a youtube video.

6.1 Class YouTubeScraper

Class for communicating with the gdata youtube API.

6.1.1 Methods

___init___(*self*)

Set the gdata youtube urls and the logger.

fetch_comments(*self*, *video_id*, *number*=0)

```
fetch a number of youtube comments using _comment_generator.

Parameters:
- video_id : the id of the youtube video
- number : the number of comments to fetch (0 = all comments)

Returns:
- list of Comment objects
```

fetch_videoinfo(*self*, *video_id*)

```
fetch relevant information about the video from the gdata youtube API.

Parameters:
- video_id : the id of the youtube video

Returns:
- tuple of Video object and list of Category objects
```

extract_categories(*self*, *req*, *video_id*)

```
extract categories from a json-converted gdata video HTTP response.

Parameters:
- req : the gdata video HTTP response
- video_id: the youtube video id
Returns:
- list of Category objects
```

extract_video(*self*, *req*, *video_id*)

```
extract video object from a json-converted gdata video HTTP response.

Parameters:
- req: the gdata video HTTP response
- video_id: the youtube video id
Returns:
- a Video object
```

7 Package test

init py-file for test.

7.1 Modules

- **test_codeformat**: This module contains tests of code formats.
  (Section 8, p. 15)
- **test_flask**: Module for integration testing the webserve module.
  (Section 9, p. 17)
- **test_sentiment_analysis**: Tests for the module sentiment_analysis.
  (Section 10, p. 21)
- **test_youtube**: tests for the youtube module.
  (Section 11, p. 22)

7.2 Variables

Name	Description
___package___	**Value:** `None`

8 Module test.test_codeformat

This module contains tests of code formats.

- Flake8
- Unittest
- Pylint

  The Unittest is defined individually in modules pr. modules wanted to be tested

8.1 Variables

Name	Description
CWD	**Value:** `'/home/syre/Dropbox/opgaver/Kandidat/02819 Data mining me...`
___package___	**Value:** `'test'`

8.2 Class TestCodeFormat

object

unittest.case.TestCase

test.test_codeformat.TestCodeFormat

Creating, listing and running tests.

8.2.1 Methods

test_pylint_compliance(*self*)

Test the modules for pylint violations.

test_flake8_compliance(*self*)

Test the modules for flake8 violations.

test_pep257_compliance(*self*)

Test the modules for pep257 violations.

Inherited from unittest.case.TestCase

___call___(), ___eq___(), ___hash___(), ___init___(), ___ne___(), ___repr___(), ___str___(), addCleanup(), addTypeEqualityFunc(), assertAlmostEqual(), assertAlmostEquals(), assertDictContainsSubset(), assertDictEqual(), assertEqual(), assertEquals(), assertFalse(), assertGreater(), assertGreaterEqual(), assertIn(), as-

sertIs(), assertIsInstance(), assertIsNone(), assertIsNot(), assertIsNotNone(), assertItemsEqual(), assertLess(), assertLessEqual(), assertListEqual(), assertMultiLineEqual(), assertNotAlmostEqual(), assertNotAlmostEquals(), assertNotEqual(), assertNotEquals(), assertNotIn(), assertNotIsInstance(), assertNotRegexpMatches(), assertRaises(), assertRaisesRegexp(), assertRegexpMatches(), assertSequenceEqual(), assertSetEqual(), assertTrue(), assertTupleEqual(), assert_(), countTestCases(), debug(), defaultTestResult(), doCleanups(), fail(), failIf(), failIfAlmostEqual(), failIfEqual(), failUnless(), failUnlessAlmostEqual(), failUnlessEqual(), failUnlessRaises(), id(), run(), setUp(), setUpClass(), shortDescription(), skipTest(), tearDown(), tearDownClass()

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

8.2.2 Properties

Name	Description
Inherited from object	
___class___	

8.2.3 Class Variables

Name	Description
Inherited from unittest.case.TestCase	
longMessage, maxDiff	

9 Module test.test_flask

Module for integration testing the webserve module.

9.1 Functions

insert_rows(*video_ids*=None, *positive_list*=None)

Helper function for inserting test rows in the database.

9.2 Class WebServeTestCase

unittest.TestCase
└ **test.test_flask.WebServeTestCase**

Class to test webserve module.

9.2.1 Methods

setUp(*self*)

setUp method for all tests. set up method, running before each test, sets up an in-memory sqlite database for use as test database and sets flask up for testing

tearDown(*self*)

tearDown method for all tests. tear down method, running after each test, closes the session

test_start_page_load_correct(*self*)

Test that the start page is loading correctly.

asserts on text in index page

test_video_page_load_correct_from_database(*self*)

Test that video loads from database directly if found.

asserts on text on video analysis page

test_video_page_load_error_wrong_id(*self*)

Test that tries to input an invalid video id at the start page.

asserts on error text in video analysis page

test_video_page_load_correct_full_youtubeurl(*self*)

Test of video page with URL.

test that loads video page when given a full youtube url like: "https://www.youtube.com/watch?v=tkXr3uxM2fY" asserts on text in video analysis page

test_video_page_load_correct_from_youtube(*self*)

Test that fetches youtube information and loads video page.

this can be considered the "normal use case" asserts on text in video analysis page

test_video_page_saves_video_in_db(*self*)

Test asserting video is saved in database after video page load.

asserts on test database query

test_video_page_updates_sentiment_in_db(*self*)

Test for "outdated" video in database.

testing if sentiment is updated in the database if a video previously saved in database is updated at youtube (contains new comments) asserts on test database query

test_video_page_saves_comment_in_db(*self*)

Test asserting comments are saved after video page load.

asserts on test database query

test_video_page_saves_commentsentiment_in_db(*self*)

Test asserting comment sentiments are saved after video page load.

asserts on test database query

test_video_page_saves_videosentiment_in_db(*self*)

Test asserting a videosentiment is saved after video page load.

asserts on test database query

test_video_page_comment_sentiment_plot_only_negative(*self*)

Test asserting the comment sentiment plot works (with negative).

tests with only negative comment sentiments

test_video_page_comment_sentiment_plot_only_positive(*self*)

Test asserting the comment sentiment plot works (with positive).

tests with only positive comment sentiments

test_video_page_comment_sentiment_plot_mixed(*self*)

Test asserting the comment sentiment plot works (mixed).

with mixed comment sentiments (positive and negative)

test_video_page_video_sentiment_plot_correct(*self*)

Test that video sentiment on video page load works correctly.

asserts on HTTP status = 200

test_previous_page_taking_newest(*self*)

Test that the previous page shows the 5 most recent analyses.

inserts 10 test result and asserts on 5 last ids

test_about_page_load_correct(*self*)

Test about page loads correctly.

asserts on text on about page

test_error_page_load_from_wrong_url(*self*)

Test that webservice fails gracefully on wrong url.

ensures an appropriate response is returned when trying to load a page that does not exist

test_video_page_error_disallowed_comments_video(*self*)

Test edge-case: video with comments disallowed.

Test that ensures an appropriate response is returned when trying to analyze a video with comments disabled

test_video_page_error_no_comments_video(*self*)

Test edge-case: video with no comments.

Test that ensures an appropriate response is returned when trying to analyze a video with no comments

asserts on error text

10 Module test.test_sentiment_analysis

Tests for the module sentiment_analysis.

10.1 Class SentimentAnalysisTestCase

unittest.TestCase
┐
test.test_sentiment_analysis.SentimentAnalysisTestCase

This class has test-methods for sentiment_analysis module.

10.1.1 Methods

test_load_classifier(*self*, *train*, *load_classifier*, *load_data*)

Test the load_classifier method.

:param train: :param load_classifier: :param load_data: :return:

test_classify_comments(*self*)

Test the classify_comment method in sentiment_analysis.

test_eval(*self*)

Test the eval method in sentiment_analysis.

test_load_wrong_file(*self*, *nltk_load*, *train*, *logger*)

```
Test load method.

tests with wrong file-name
(or the file doesn't exist)

:param nltk_load: Mock object on nltk.load method with side_effect
:param train: Mock object for train method. The method is not been
              called
:param logger: Mock object on logging method
```

11 Module test.test_youtube

tests for the youtube module.

11.1 Class YouTubeTestCase

unittest.TestCase
└ **test.test_youtube.YouTubeTestCase**

This class has test-methods for youtube module.

11.1.1 Methods

test_fetch_comments_correct_id(*self*, *mock_comment*)

```
test of fetch_comments in YouTubeScraper with correct id.

calling the the fetch_comments method in the YouTubeScraper with
 a correct id and asserting the results
:param mock_comment: Mock object for comment method. The method is not
                     been called
```

test_fetch_comments_returns_correct_over_zero(*self*, *mock_comment*)

test of fetch_comments returning all comments correctly.

this method tests that the correct amount of comments is returned when specified :param mock: Mock object for _comment_generator

test_fetch_comments_returns_all_at_zero(*self*, *mock_comment*)

test fetch_comments returning all comments unexplicitly.

tests that all comments are returned when a number is not explicitly specified :param mock_comment: Mock object for _comment_generator

test_comment_generator_wrong_videoid_gracefully(*self*, *mock_logger*)

test _comment_generator handles wrong video_id.

tests that _comment_generator raises a ValueError when supplied with an invalid video id :param mock_logger: Mock object for logger

test_fetch_videoinfo_wrong_videoid_gracefully(*self*, *mock_logger*)

test fetch_videoinfo handles wrong video_id.

tests that fetch_videoinfo raises exception and error is logged when supplied with an invalid video id :param mock_logger: Mock object for logger

test_fetchcomments_no_connection(*self*, *mock_requests*, *mock_logger*)

test in case of requests error.

tests that connection error (requests) is logged :param mock_requests : Mock object for requests.get method :param mock_logger : Mock object for logger

Index