

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-processed 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 possibilities: 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>

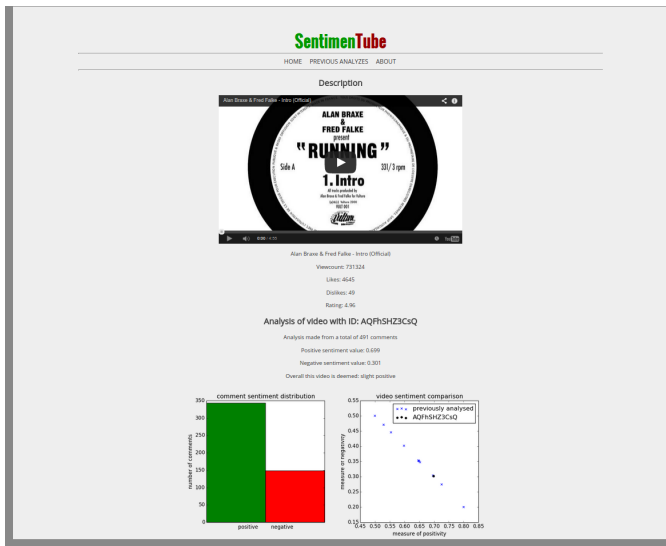


Figure 1. Result of sentiment analysis of YouTube video

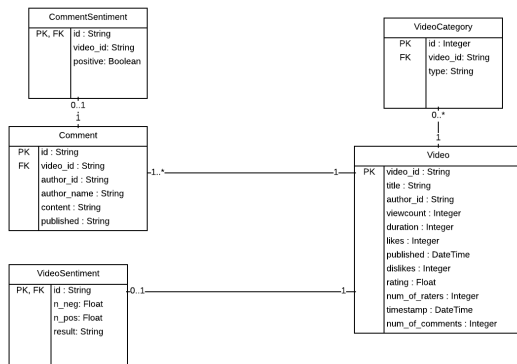


Figure 2. Database schema

APPENDIX A
CODE LISTINGS
LISTINGS

../sentimentube/youtube.py	4
../sentimentube/sentiment_analysis.py	6
../sentimentube/webserve.py	9
../sentimentube/models.py	12
../sentimentube/database.py	13
../test/test_flask.py	14
../test/test_youtube.py	18
../test/test_sentiment_analysis.py	19
../test/test_codeformat.py	20

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # pylint: disable=R0201
4  """_This module scrapes /download_contents_from_a_youtube_video._"""
5  import requests
6  import dateutil.parser
7  import logging
8  import datetime
9
10 import models
11
12
13 class YouTubeScraper:
14
15     """_Class for communicating with the gdata.youtube.API._"""
16
17     def __init__(self):
18         """_Set the gdata.youtube.urls and the logger._"""
19         self.comment_url = "https://gdata.youtube.com/feeds/api/videos/{0}/" \
20             "comments"
21         self.video_url = "https://gdata.youtube.com/feeds/api/videos/{0}"
22         self.logger = logging.getLogger(__name__)
23
24     def _comment_generator(self, video_id):
25         """
26         _generator for fetching one page of youtube comments.
27
28         _For a youtube video, it returns a list of comment dictionaries
29         with keys: author_name, author_id, content, video_id, id, published
30         _It should not be used directly, it is private and the fetch_comments
31         _method should be used instead.
32
33         Parameters:
34         _video_id: the id of the youtube video
35         """
36         next_url = self.comment_url.format(video_id)
37         params = {"v": 2, "alt": "json", "max-results": 50,
38                 "orderby": "published"}
39
40         while True:
41             if next_url:
42                 try:
43                     response = requests.get(next_url, params=params)
44                 except requests.exceptions.RequestException:
45                     self.logger.exception("_comment_generator: request failed")
46                     raise
47             else:
48                 if not response:
49                     error = "_comment_generator: invalid video id: " \
50                         "{}".format(video_id)
51                     self.logger.error(error)
52                     raise ValueError(error)
53                 response = response.json()
54             else:
55                 raise StopIteration
56             comments = []
57             if "entry" not in response["feed"]:
58                 raise RuntimeError("no comments for video")
59             for comment in response["feed"]["entry"]:
60                 author_name = comment["author"][0]["name"]["$t"]
61                 author_id = comment["author"][0]["yt$userId"]["$t"]
62                 content = comment["content"]["$t"]
63                 comment_id = comment["id"]["$t"]
64                 published = dateutil.parser.parse(comment["published"]["$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 a 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$accessControl"] if
120                              entry["action"] == "comment"][0]
121
122         if comment_permission == "denied":
123             self.logger.error("fetch_videoinfo: comments disallowed for video")
124
125             raise RuntimeError("Comments disallowed for video-{0}".format(
126                 video_id))
127
128         video = self.extract_video(req, video_id)
129         categories = self.extract_categories(req, video_id)
130         return video, categories
131
132     def extract_categories(self, req, video_id):
133         """
134         Extract categories from a json-converted gdata video HTTP response.
135
136         Parameters:
137         req: the gdata video HTTP response
138         video_id: the youtube video id
139
140         Returns:
141         list of Category objects
142         """
143         categories = req["entry"]["media$group"]["media$category"]
144         categories = [models.VideoCategory(type=category["$t"],
145                                           video_id=video_id)
146                       for category in categories]
147         return categories

```

```

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
187         return video
188
189 if __name__ == '__main__':
190     YOU = YouTubeScraper()
191     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):

```

```

40     """
41     """Create a list of tuples containing words of the text and its sentiment.
42
43     """param tuples: Tuples with text (as strings) and its sentiment
44     """return tuples_text: The list of tuples
45     """
46     tuple_text = []
47     stop = stopwords.words('english')
48     for (text, sentiment) in tuples:
49         words = text.split()
50         clean_word = ([i.lower() for i in words
51                        if not i.lower() in stop])
52         tuple_text.append((clean_word, sentiment))
53     return tuple_text
54
55
56 class SentimentAnalysis:
57
58     """Class for making sentiment analysis of video comments."""
59
60     def __init__(self, file_name):
61         """Call the load method to load the classifier from file."""
62         corpus_path = "data/corpus.txt"
63         self.logger = logging.getLogger(__name__)
64         self.word_list = self.create_words_and_tuples(corpus_path)
65         self.file_path = os.path.join(os.path.dirname(__file__), file_name)
66         self.classifier = self.load_classifier(corpus_path)
67
68     def load_corpus(self, file_name, split=","):
69         """
70         """Load corpus from file and stores it in a tuple.
71
72         """param file_name: Name of the corpus file
73         """param split: How to split a line in the corpus (text vs. sentiment).
74         """Default split: ','
75         """return: tuple and int: Respectively positive and negative tuple
76         """(text, sentiment)
77         """
78         self.logger.debug("Loading corpus file")
79         file_path = os.path.join(os.path.dirname(__file__), file_name)
80         try:
81             with open(file_path, 'r') as read_file:
82                 lines = read_file.readlines()[1:]
83         except (FileExistsError, LookupError):
84             self.logger.error("I/O error: corpus file not found")
85             raise
86         else:
87             sentiment_dict = {0: "negative", 1: "positive"}
88             return [(line.split(split)[1].strip(),
89                     sentiment_dict[int(line.split(split)[0])])
90                    for line in lines]
91
92     def _word_feats_extractor(self, doc):
93         """
94         """Extract features from corpus.
95
96         """param words: List of words from corpus
97         """return: Dict of the words as keys and True/False as values
98         """
99         doc_words = set(doc)
100         return dict([(("contains({})".format(i), i in doc_words)
101                       for i in self.word_list)]
102
103     def create_words_and_tuples(self, corpus_filename):
104         """
105         """Load corpus and create tagged text and word list.
106
107         """tagged_text is the words of the text and their sentiment,
108         """word_list is the words in the corpus
109
110         """param corpus_filename: the filepath of the corpus
111         """
112         text = self.load_corpus(corpus_filename, ",")
113         tagged_text = create_tagged_text(text)
114         word_list = create_word_list(tagged_text)
115         return tagged_text, word_list
116
117     def _train(self, corpus_filename):
118         """
119         """Train the classifier.
120
121         """Training the Naïve Bayes classifier, by calling the following methods:
122         """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 and 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

```

```

45     for comment_sentiment in comments_sentiment:
46         if comment_sentiment.id not in db_comment_sentiment_ids:
47             database.DB_SESSION.add(comment_sentiment)
48
49     db_videosentiment = database.DB_SESSION.query(
50         models.VideoSentiment).filter(
51             models.VideoSentiment.id == video_sentiment.id).first()
52
53     if db_videosentiment:
54         db_videosentiment = database.DB_SESSION.merge(video_sentiment)
55     else:
56         database.DB_SESSION.add(video_sentiment)
57     database.DB_SESSION.commit()
58
59
60 @APP.route("/")
61 def index():
62     """
63     Show the front page to the user.
64
65     return: the front page (index.html)
66     """
67     return flask.render_template("index.html")
68
69
70 @APP.route("/about")
71 def about():
72     """
73     Show the about page to the user.
74
75     return: the about page (about.html)
76     """
77     return flask.render_template("about.html")
78
79
80 @APP.route("/video")
81 def video():
82     """
83     Video analysis page.
84
85     Run the classification for the input the user has given
86     Checks in database whether the video has been processed before. If it has
87     been processed before and there is no changes, it simply shows the result.
88     Else, it will process the video and show the result
89     return: The video page (video.html) with the result from database or
90     classification.
91     """
92     video_id = flask.request.args.get("video_id")
93     # if in the form of an url, extract id
94     if "youtube" in video_id:
95         url = urllib.parse.urlparse(video_id)
96         query = dict(urllib.parse.parse_qs(url[4]))
97         video_id = query["v"]
98
99     db_video_info = database.DB_SESSION.query(models.Video).filter(
100         models.Video.id == video_id).first()
101
102     try:
103         video_info, categories = SCRAPER.fetch_videoinfo(video_id)
104     except ValueError:
105         return flask.render_template("error.html",
106                                     error="invalid_video_id")
107     except RuntimeError as err:
108         return flask.render_template("error.html", error=str(err))
109
110     if (db_video_info and
111         db_video_info.num_of_comments == video_info.num_of_comments):
112         LOGGGER.info("sentiment for video with id: %r found in database",
113                     video_id)
114
115         sentiment = database.DB_SESSION.query(models.VideoSentiment).filter(
116             models.VideoSentiment.id == video_id).first()
117
118         comments = database.DB_SESSION.query(models.Comment).filter(
119             models.Comment.video_id == video_id).all()
120     else:
121         LOGGGER.info("processing new video with id: %r", video_id)
122         if db_video_info:
123             db_video_info = database.DB_SESSION.merge(video_info)
124         else:
125             database.DB_SESSION.add(video_info)
126             database.DB_SESSION.add_all(categories)
127     try:
128         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.

```



```

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

```

```

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():

```

```

19 """_Create_the_database_and_its_tables_"""
20 import models # noqa # pylint: disable=unused-variable
21 BASE.metadata.create_all(bind=ENGINE)

```

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 # pylint: disable=R0904
4
5 """_Module_for_integration_testing_the_webserve_module_"""
6
7 from unittest import TestCase
8 import webserve
9 import database
10 import sqlalchemy
11 import models
12 import datetime
13
14
15 def insert_rows(video_ids=None, positive_list=None):
16     """_Helper_function_for_inserting_test_rows_in_the_database_"""
17     if not video_ids:
18         video_ids = ["tkXr3uxM2fy"]
19     if not positive_list:
20         positive_list = [True]
21     for video_index, v_id in enumerate(video_ids):
22         now = datetime.datetime.now()
23         database.DB_SESSION.add(models.Video(id=v_id,
24                                             title="test_title_{}".format(v_id),
25                                             author_id="test_author_{}".format(v_id),
26                                             viewcount=1,
27                                             duration=10,
28                                             likes=1,
29                                             published=now,
30                                             dislikes=1,
31                                             rating=3,
32                                             num_of_raters=5,
33                                             num_of_comments=5,
34                                             timestamp=now))
35         database.DB_SESSION.add(models.VideoSentiment(id=v_id,
36                                                       n_pos=5.2,
37                                                       n_neg=10.2,
38                                                       result="negative"))
39     for com_index, pos in enumerate(positive_list):
40         database.DB_SESSION.add(models.Comment(id="comment_{}_{}".format(video_index,
41                                     com_index),
42                                               video_id=v_id,
43                                               author_id="test_author_{}_id_{}".format(v_id,
44                                                 author_name="test_{}_author_{}_name_{}".format(v_id,
45                                                         content="test_comment_{}_text_{}".format(v_id,
46                                                         published=now))
47                                               .format(v_id),
48                                               author_name="test_{}_author_{}_name_{}".format(v_id,
49                                                         content="test_comment_{}_text_{}".format(v_id,
50                                                         published=now))
51                                               .format(v_id),
52                                               content="test_comment_{}_text_{}".format(v_id,
53                                                         published=now))
54                                               .format(v_id),
55                                               content="test_comment_{}_text_{}".format(v_id,
56                                                         published=now))
56         database.DB_SESSION.add(
57             models.CommentSentiment(
58                 id="comment_{}_{}".format(video_index,
59                                           com_index),
60                 video_id=v_id, positive=pos))
61     database.DB_SESSION.commit()
62
63
64
65
66 class WebServeTestCase(TestCase):
67     """_Class_to_test_webserve_module_"""
68     def setUp(self):
69         """
70         =====setUp_method_for_all_tests.
71         =====setUp_method,_running_before
72         =====each_test,_sets_up_an_in-memory_sqlite_database
73         =====for_use_as_test_database_and
74         =====sets_flask_up_for_testing
75         =====
76         """
77         webserve.APP.config["TESTING"] = True
78         self.app = webserve.APP.test_client()
79         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         .....tearDown 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
197         database.DB_SESSION.commit()
198
199         self.app.get("/video?video_id={}".format(v_id))
200
201         sentiment = database.DB_SESSION.query(
202             models.VideoSentiment).filter_by(id=v_id).first()
203         assert sentiment.n_neg != negative_score
204         assert sentiment.n_pos != positive_score
205         assert sentiment.result != "test_positive"
206
207     def test_video_page_saves_comment_in_db(self):
208         """
209         Test asserting comments are saved after video page load.
210
211         asserts on test database query
212         """
213         v_id = "tkXr3uxM2fY"
214         self.app.get("/video?video_id={}".format(v_id))
215         comment = database.DB_SESSION.query(
216             models.Comment).filter_by(video_id=v_id).first()
217         assert comment
218
219     def test_video_page_saves_commentsentiment_in_db(self):
220         """
221         Test asserting comment sentiments are saved after video page load.
222
223         asserts on test database query
224         """
225         v_id = "tkXr3uxM2fY"
226         self.app.get("/video?video_id={}".format(v_id))
227         sentiment = database.DB_SESSION.query(
228             models.CommentSentiment).filter_by(video_id=v_id).first()
229         assert sentiment
230
231     def test_video_page_saves_videosentiment_in_db(self):
232         """
233         Test asserting a videosentiment is saved after video page load.
234
235         asserts on test database query
236         """
237         v_id = "tkXr3uxM2fY"
238         self.app.get("/video?video_id={}".format(v_id))
239         sentiment = database.DB_SESSION.query(
240             models.VideoSentiment).filter_by(id=v_id).first()
241         assert sentiment
242
243     def test_video_page_comment_sentiment_plot_only_negative(self):
244         """
245         Test asserting the comment sentiment plot works (with negative).

```



```

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", "zNJBD_I5EU",
299             "v2zTVZFICZ0", "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 Søren 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):

```

```

329         """
330         """Test edge-case: video with comments disallowed.
331
332         """Test that ensures an appropriate response is returned
333         """when trying to analyze a video with comments disabled
334         """
335         response = self.app.get("/video?video_id={}".format("NZQQdlPoz5g"))
336         assert "Error: Comments disallowed for video" in \
337             response.data.decode("utf-8")
338
339         def test_video_page_error_no_comments_video(self):
340             """
341             """Test edge-case: video with no comments.
342
343             """Test that ensures an appropriate response is returned
344             """when trying to analyze a video with no comments
345
346             """asserts on error text
347             """
348             response = self.app.get("/video?video_id={}".format("wv4ol_Q4G_k"))
349             assert "Error: no comments for video" in \
350                 response.data.decode("utf-8")

```

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # pylint: disable=W0212, R0201
5
6  """_tests_for_the_youtube_module._"""
7
8  from unittest import mock, TestCase
9  import youtube
10 import models
11 import requests
12
13
14 class YouTubeTestCase(TestCase):
15
16     """_This class has test-methods_for_youtube_module._"""
17
18     @mock.patch("youtube.YouTubeScraper._comment_generator")
19     def test_fetch_comments_correct_id(self, mock_comment):
20         """
21         """test_of_fetch_comments_in_YouTubeScraper_with_correct_id.
22
23         """calling_the_fetch_comments_method_in_the_YouTubeScraper_with
24         """a_correct_id_and_asserting_the_results
25         """param_mock_comment: Mock object for comment method. The method is not
26         """been called
27         """
28         scraper = youtube.YouTubeScraper()
29         # return dummy list
30         cm = models.Comment(id="test", video_id="test", author_id="test",
31                             author_name="test", content="test",
32                             published="test")
33         mock_comment.return_value = iter([cm for x in range(100)])
34         scraper.fetch_comments("dQw4w9WgXcQ", 50)
35         scraper._comment_generator.assert_called_with("dQw4w9WgXcQ")
36
37         @mock.patch("youtube.YouTubeScraper._comment_generator")
38         def test_fetch_comments_returns_correct_over_zero(self, mock_comment):
39             """
40             """test_of_fetch_comments_returning_all_comments_correctly.
41
42             """this_method_tests_that_the_correct_amount_of_comments_is_returned
43             """when_specified
44             """param_mock: Mock object for comment_generator
45             """
46             scraper = youtube.YouTubeScraper()
47             # return dummy list
48             cm = models.Comment(id="test", video_id="test", author_id="test",
49                                 author_name="test", content="test",
50                                 published="test")
51             mock_comment.return_value = iter([cm for x in range(500)])
52             comments = scraper.fetch_comments("dQw4w9WgXcQ", 250)
53             assert len(comments) == 250
54
55             @mock.patch("youtube.YouTubeScraper._comment_generator")
56             def test_fetch_comments_returns_all_at_zero(self, mock_comment):
57                 """
58                 """test_fetch_comments_returning_all_comments_unexplicitly.
59
60                 """tests_that_all_comments_are_returned_when_a_number

```

```

61 .....is_not_explicitly_specified
62 .....:param_mock_comment:_Mock_object_for_comment_generator
63 .....
64         scraper = youtube.YouTubeScraper()
65         # return dummy list
66         cm = models.Comment(id="test", video_id="test", author_id="test",
67                             author_name="test", content="test",
68                             published="test")
69         mock_comment.return_value = iter([[cm for x in range(500)]])
70         comments = scraper.fetch_comments("dQw4w9WgXcQ")
71         assert len(comments) == 500
72
73         @mock.patch("logging.Logger.error")
74         def test_comment_generator_wrong_video_id_gracefully(self, mock_logger):
75             """
76             .....test_comment_generator_handles_wrong_video_id.
77
78             .....tests_that_comment_generator_raises_a_ValueError
79             .....when_supplied_with_an_invalid_video_id
80             .....:param_mock_logger:_Mock_object_for_logger
81             .....
82             scraper = youtube.YouTubeScraper()
83             generator = scraper._comment_generator("wrong_url")
84             self.assertRaises(ValueError, next, generator)
85             mock_logger.assert_called()
86
87             @mock.patch("logging.Logger.error")
88             def test_fetch_videoinfo_wrong_video_id_gracefully(self, mock_logger):
89                 """
90                 .....test_fetch_videoinfo_handles_wrong_video_id.
91
92                 .....tests_that_fetch_videoinfo_raises_exception
93                 .....and_error_is_logged_when_supplied_with_an_invalid_video_id
94                 .....:param_mock_logger:_Mock_object_for_logger
95                 .....
96                 scraper = youtube.YouTubeScraper()
97                 self.assertRaises(ValueError, scraper.fetch_videoinfo, "wrong_url")
98                 mock_logger.assert_called()
99
100             @mock.patch("logging.Logger.exception")
101             @mock.patch("requests.get")
102             def test_fetchcomments_no_connection(self, mock_requests, mock_logger):
103                 """
104                 .....test_in_case_of_requests_error.
105
106                 .....tests_that_connection_error_(requests)_is_logged
107                 .....:param_mock_requests:_Mock_object_for_requests.get_method
108                 .....:param_mock_logger:_Mock_object_for_logger
109                 .....
110                 scraper = youtube.YouTubeScraper()
111                 mock_requests.side_effect = requests.exceptions.RequestException
112                 assert mock_logger.assert_called()
113                 self.assertRaises(requests.exceptions.RequestException,
114                                 scraper.fetch_comments, "dQw4w9WgXcQ")

```

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # pylint: disable=W0212, R0201
5
6  """_Tests_for_the_module_sentiment_analysis._"""
7  from unittest import mock, TestCase
8  import sentiment_analysis
9  import models
10 from datetime import datetime
11
12
13 class SentimentAnalysisTestCase(TestCase):
14
15     """_This_class_has_test_methods_for_sentiment_analysis_module._"""
16
17     @mock.patch("sentiment_analysis.SentimentAnalysis._train")
18     @mock.patch("sentiment_analysis.SentimentAnalysis.load_classifier")
19     @mock.patch("nltk.data.load")
20     def test_load_classifier(self, train, load_classifier, load_data):
21         """
22         .....Test_the_load_classifier_method.
23
24         .....:param_train:
25         .....:param_load_classifier:
26         .....:param_load_data:
27         .....:return:
28         .....

```

```

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("nlk.data.load")
84         @mock.patch("sentiment_analysis.SentimentAnalysis._train")
85         @mock.patch("logging.Logger")
86         def test_load_wrong_file(self, nlk_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             nlk_load.side_effect = (FileExistsError, LookupError)
99
100             sentiment_analysis.SentimentAnalysis(
101                 "data/hello_hello.pickle")
102
103             logger.assert_called()
104             self.assertRaises(FileExistsError, nlk_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

Demonstration using epydoc:

```
epydoc --pdf -o /home/fnielsen/tmp/epydoc/ --name RBBase wikipedia/api.py
```

Test output (nosetests):

```
nose.config: INFO: Ignoring files matching ['^\\.\\.py$', '^_\\.py$', '^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