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(), assertIs(), 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