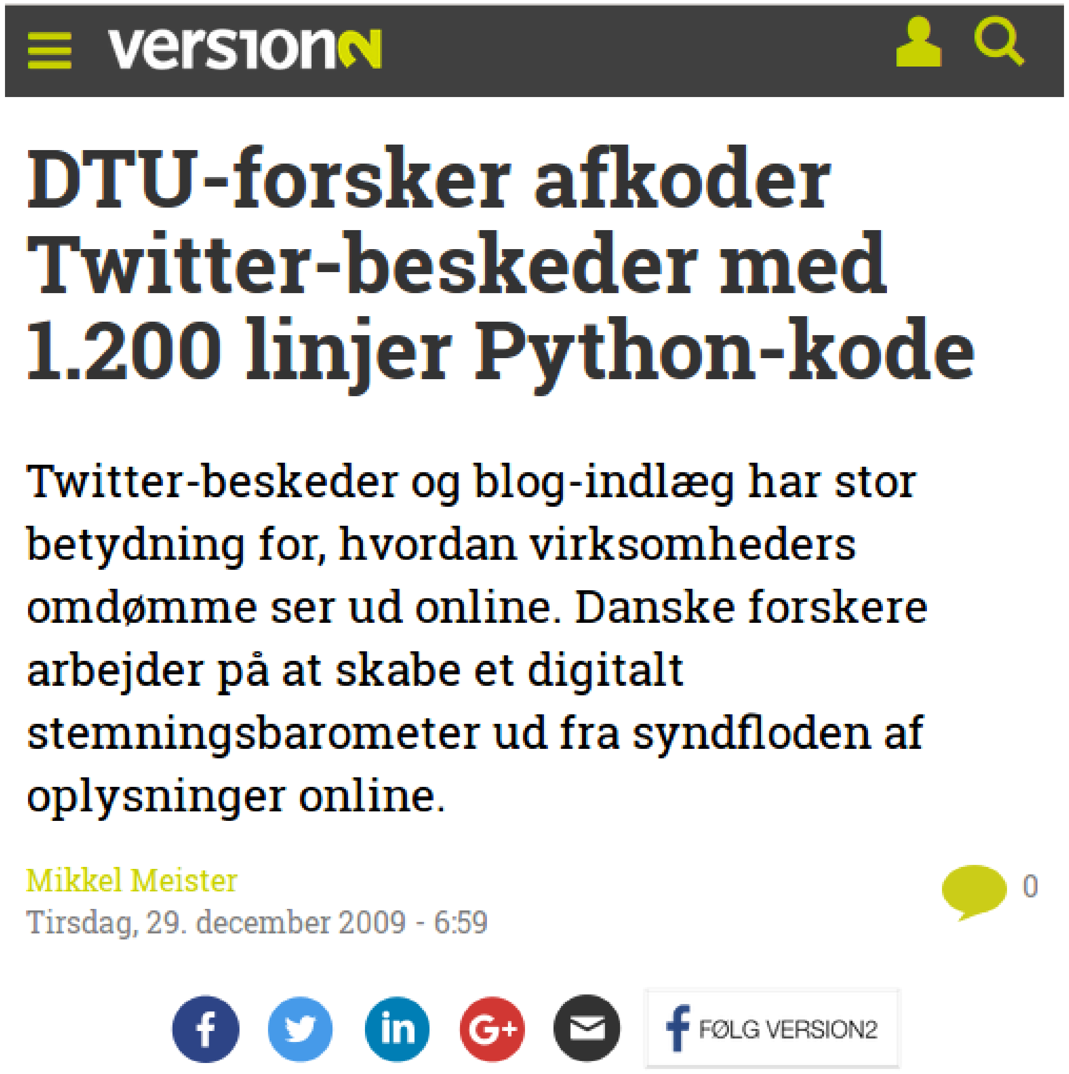
# afinn project

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

March 28, 2017

## afinn



# DTU-forsker afkoder Twitter-beskeder med 1.200 linjer Python-kode

Twitter-beskeder og blog-indlæg har stor betydning for, hvordan virksomheders omdømme ser ud online. Danske forskere arbejder på at skabe et digitalt stemningsbarometer ud fra syndfloden af oplysninger online.

Mikkel Meister
Tirsdag, 29. december 2009 - 6:59

Started out as a English sentiment word list for use in analysis of Twitter messages in 2009.

Later the approach was evaluated with manually labeled tweets in published paper.

Shown Python code snippets on the Internet including my blog on how to use it.

In July 2015, turned into a GitHub repository.

0.1 release in November 2016.

## Philosophies for `afinn`

Simple approach with little dependencies: The package should do what it should do and nothing more.

Open source.

Test thoroughly all elements of the package.

Documentation in the code for everything.

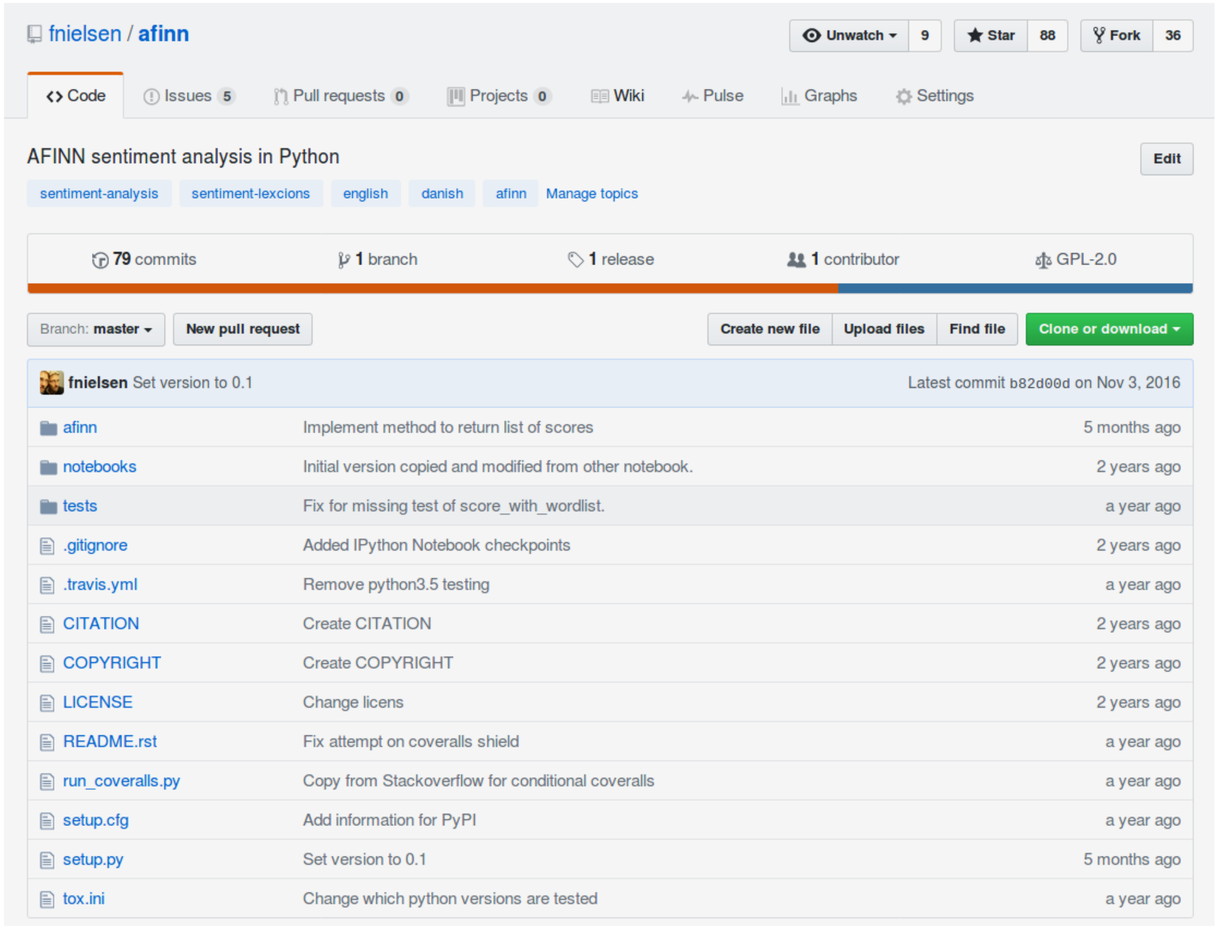
Tutorials.

Easy installation for other developers.

Should work for a broad number of Python versions.

"Python best practice".

## GitHub-based development


fnielsen / afinn

AFINN sentiment analysis in Python

sentiment-analysis · sentiment-lexcions · english · danish · afinn · Manage topics

79 commits · 1 branch · 1 release · 1 contributor · GPL-2.0

fnielsen Set version to 0.1 — Latest commit b82d00d on Nov 3, 2016

| | | |
|---|---|---|
| afinn | Implement method to return list of scores | 5 months ago |
| notebooks | Initial version copied and modified from other notebook. | 2 years ago |
| tests | Fix for missing test of score_with_wordlist. | a year ago |
| .gitignore | Added IPython Notebook checkpoints | 2 years ago |
| .travis.yml | Remove python3.5 testing | a year ago |
| CITATION | Create CITATION | 2 years ago |
| COPYRIGHT | Create COPYRIGHT | 2 years ago |
| LICENSE | Change licens | 2 years ago |
| README.rst | Fix attempt on coveralls shield | a year ago |
| run_coveralls.py | Copy from Stackoverflow for conditional coveralls | a year ago |
| setup.cfg | Add information for PyPI | a year ago |
| setup.py | Set version to 0.1 | 5 months ago |
| tox.ini | Change which python versions are tested | a year ago |

Git-based development with GitHub.

Repository contains the Python module itself with data, test function, setup and package files files (`setup.py`, README.rst), notebooks with example code.

Other developers can work from it: 36 forks by different peoples.

## The AFINN word list

Word associated with sentiment score between −5 (most negative) and +5 (most positive):

```
abandon	-2
abandoned	-2
abandons	-2
abducted	-2
abduction	-2
abductions	-2
abhor	-3
abhorred	-3
abhorrent	-3
abhors	-3
abilities	2
ability	2
aboard	1
aborted	-1
```

## Basic `Afinn` object

The word list is encapulated as a Python class (object-orientation)

The word list is loaded at object instantiation time, to avoid reading overhead during sentiment scoring

A text scored for sentiment based on the sentiment of individual words with a method from the class:

```
class Afinn():
    def __init__(self):
        self.data = self.load_data()
    def score(self, text):
        score = 0
        for word in text:
            score += self.data.get(word, default=0)
        return score
```

## Basic use

Using the class: Object instantiation followed by calling the score methods:

```
>>> from afinn import Afinn
>>> afinn = Afinn()  # afinn is a object name now, not module
>>> afinn.score('It␣is␣so␣horrendously␣bad')
-3.0
>>> afinn.score('very␣funny')
4.0
```

Or score multiple texts in a list:

```
afinn_scores = [afinn.score(text) for text in texts]
```

## Basic processing

The central part of the text processing uses regular expression (Python module: `re`) to extract words or to directly match against the AFINN dictionary.

```
import re  # Import regular expression standard library module

# Setup
lexicon = {'ikke␣god': -2, 'imponerende': 3, 'ineffektiv': -2}
regex = re.compile('(ikke␣god|imponerende|ineffektiv)')

# Match and scoring
matched = regex.findall("Den␣er␣ineffektiv␣og␣ikke␣god")
score = sum([lexicon[word] for word in matched])
```

`score` is now $-4$. A few phrases can be matched.

## Code checking

`flake8` tool can check that the code conforms to convention (PEP8).

```
$ flake8 afinn
```

(Nothing is reported if there is no convention issues)

Further checking can be made with `pylint`.

## Documentation

Documention in the “docstring” of a object method:

```
def scores_with_pattern(self, text):
    """Score text based on pattern matching.

    Performs the actual sentiment analysis on a text. It uses a regular
    expression match against the word list.

    The output is a list of float variables for each matched word or
    phrase in the word list.

    Parameters
    ----------
    text : str
        Text to be analyzed for sentiment.

    Returns
    -------
    scores : list of floats
        Sentiment analysis scores for text
```

## Documentation

and the documentation goes on with example code:

```
Examples
--------
>>> afinn = Afinn()
>>> afinn.scores_with_pattern('Good␣and␣bad')
[3, -3]

>>> afinn.scores_with_pattern('some␣kind␣of␣idiot')
[0, -3]

"""
# TODO: ":D" is not matched
words = self.find_all(text)
scores = [self._dict[word] for word in words]
return scores
```

15 lines of documentation, 3 lines of code.

## Documention checking

There is a standard for documentation: PEP 257.

Tools exists to check whether the documentation is complete and whether it follows the standard: `pydocstyle` (previously called `pep257`).

I can call it with:

```
pydocstyle afinn
```

(It should report nothing if ok)

There is a plugin in `flake8`

Afinn uses the Numpy document convention. However this cannot be tested: Currently no tools (AFAIK).

## Testing

Unit tests in `afinn/tests/test_afinn.py`

Test function have the prefix `test_`.

The prefix tells py.test, http://doc.pytest.org, to test it.

Example for testing the `find_all` method of the object:

```
def test_find_all():
    afinn = Afinn()
    words = afinn.find_all("It␣is␣so␣bad")
    assert words == ['bad']
```

Here it is tested whether `find_all` returns a list with a single element “bad”.

## Testing

Starting `py.test` in the `afinn` directory will automatically identify all test functions that should be executed based on `test_` prefix:

```
$ py.test
================================== test session starts ================
platform linux -- Python 3.5.2, pytest-3.0.6, py-1.4.32, pluggy-0.4.0
rootdir: /home/faan/projects/afinn, inifile:
collected 14 items

tests/test_afinn.py ..............

=============================== 14 passed in 0.49 seconds =============
```

Succinct!

## Testing: doctesting

From method documentation:

```
Examples
--------
>>> afinn = Afinn()
>>> afinn.scores_with_pattern('Good␣and␣bad')
[3, -3]
```

This piece of code can be tested: “doctest”

```
python -m doctest afinn/afinn.py
```

or . . .

## Testing: doctesting

Testing the entire module:

```
$ py.test --doctest-modules afinn
======================================= test ...
platform linux -- Python 3.5.2, pytest-3.0.6, py-1.4.32, ...
rootdir: /home/faan/projects/afinn, inifile:
collected 7 items

afinn/afinn.py .......

===================================== 7 passed
```

Here 7 example code snippets were found in the docstrings, extracted and tested and found to be ok.

## Testing with tox

I would like to have `afinn` working with different versions of Python: Versions 2.6, 2.7, 3.3, 3.4 and 3.5.

`tox` combines testing with **virtual environments** enabling the test of **different versions of Python**.

`tox` creates virtual environments in `afinn/.tox/<virtualenv>` moves into them and executes whatever is specified in a `tox.ini` file (for afinn it is setup to execute py.test, doctesting and flake8).

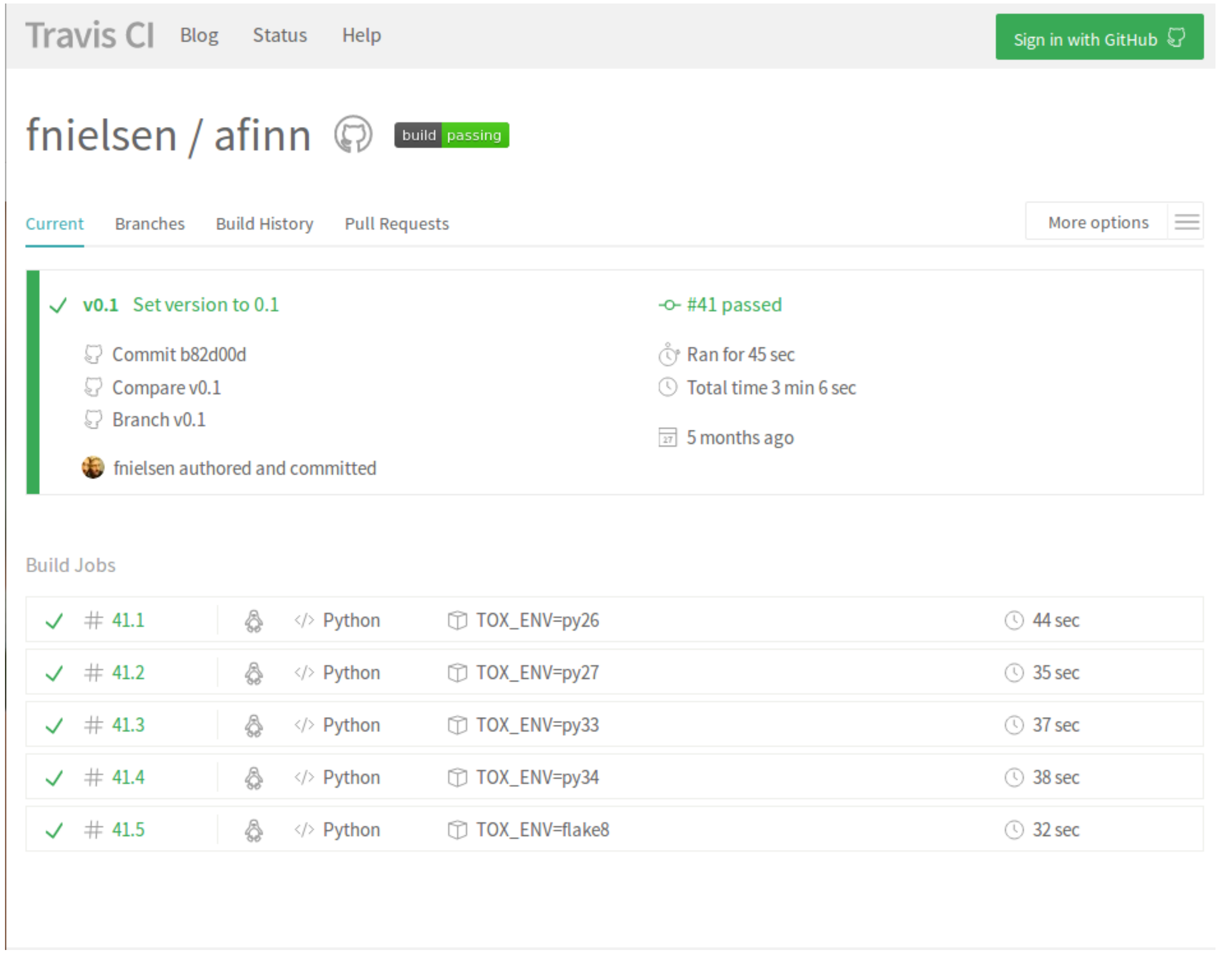
`tox` neatly enables testing multiple versions with just a single command.

## Testing with tox

```
$ tox
GLOB sdist-make: /home/faan/projects/afinn/setup.py
py26 inst-nodeps: /home/faan/projects/afinn/.tox/dist/afinn-0.1.zip
...
Installing collected packages: afinn
  Running setup.py install for afinn ... done
Successfully installed afinn-0.1
py26 runtests: commands[1] | py.test test_afinn.py
==================================== test session starts
platform linux2 -- Python 2.6.9, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: /home/faan/projects/afinn, inifile:
collected 14 items

test_afinn.py ..............
...
  py26: commands succeeded
  py27: commands succeeded
  py33: commands succeeded
  py34: commands succeeded
  py35: commands succeeded
  flake8: commands succeeded
  congratulations :)
```

## Testing with Travis



fnielsen / afinn

✓ v0.1 Set version to 0.1 — #41 passed

Commit b82d00d — Ran for 45 sec

Compare v0.1 — Total time 3 min 6 sec

Branch v0.1 — 5 months ago

fnielsen authored and committed

Build Jobs

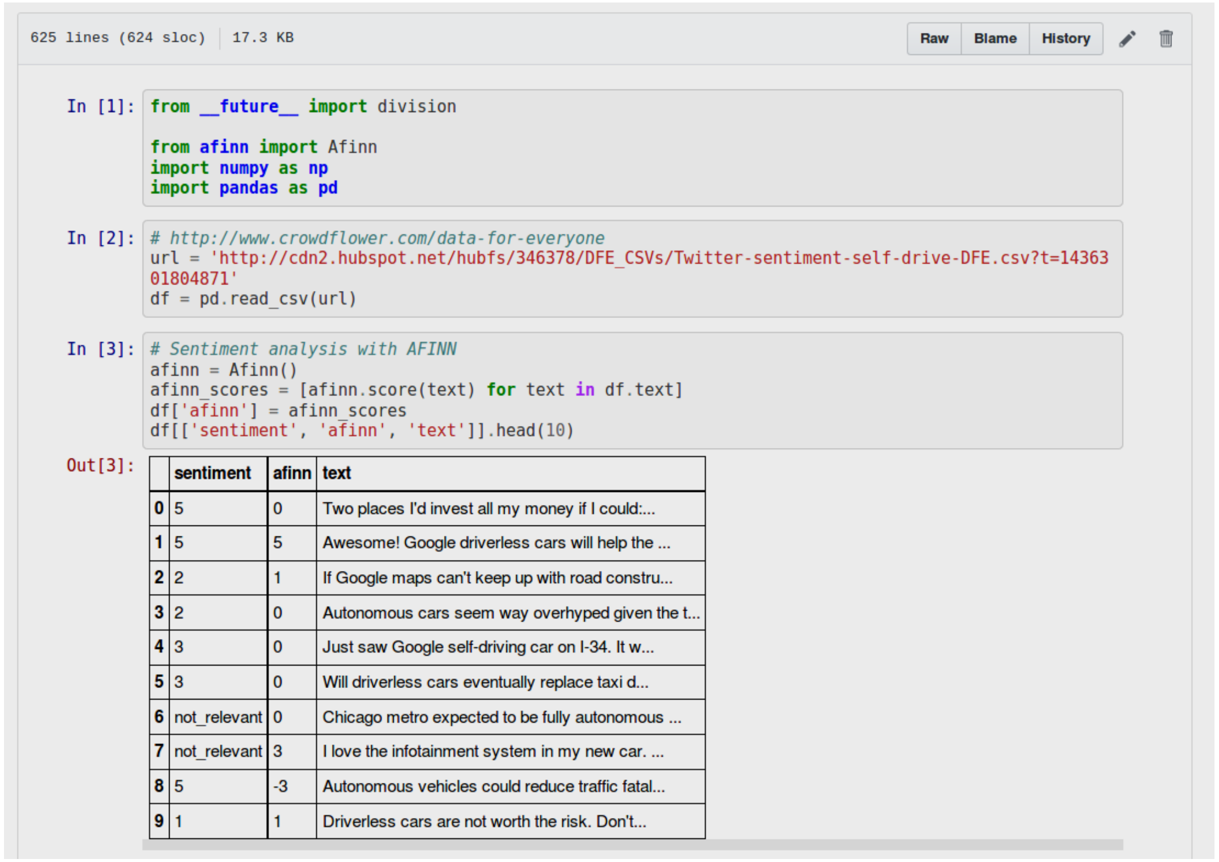
| | | | | |
|---|---|---|---|---|
| ✓ | # 41.1 | Python | TOX_ENV=py26 | 44 sec |
| ✓ | # 41.2 | Python | TOX_ENV=py27 | 35 sec |
| ✓ | # 41.3 | Python | TOX_ENV=py33 | 37 sec |
| ✓ | # 41.4 | Python | TOX_ENV=py34 | 38 sec |
| ✓ | # 41.5 | Python | TOX_ENV=flake8 | 32 sec |

Travis: cloud-based testing at `https://travis-ci.org/fnielsen/afinn`

Ensures that the package would also work on another system: Missing data? Missing dependencies?

Specified with a `.travis.yml` configuration file to run `tox`.

## Jupyter notebooks



625 lines (624 sloc) | 17.3 KB

```
In [1]: from __future__ import division

        from afinn import Afinn
        import numpy as np
        import pandas as pd

In [2]: # http://www.crowdflower.com/data-for-everyone
        url = 'http://cdn2.hubspot.net/hubfs/346378/DFE_CSVs/Twitter-sentiment-self-drive-DFE.csv?t=14363
        01804871'
        df = pd.read_csv(url)

In [3]: # Sentiment analysis with AFINN
        afinn = Afinn()
        afinn_scores = [afinn.score(text) for text in df.text]
        df['afinn'] = afinn_scores
        df[['sentiment', 'afinn', 'text']].head(10)

Out[3]:
```

| | sentiment | afinn | text |
|---|---|---|---|
| 0 | 5 | 0 | Two places I'd invest all my money if I could:... |
| 1 | 5 | 5 | Awesome! Google driverless cars will help the ... |
| 2 | 2 | 1 | If Google maps can't keep up with road constru... |
| 3 | 2 | 0 | Autonomous cars seem way overhyped given the t... |
| 4 | 3 | 0 | Just saw Google self-driving car on I-34. It w... |
| 5 | 3 | 0 | Will driverless cars eventually replace taxi d... |
| 6 | not_relevant | 0 | Chicago metro expected to be fully autonomous ... |
| 7 | not_relevant | 3 | I love the infotainment system in my new car. ... |
| 8 | 5 | -3 | Autonomous vehicles could reduce traffic fatal... |
| 9 | 1 | 1 | Driverless cars are not worth the risk. Don't... |

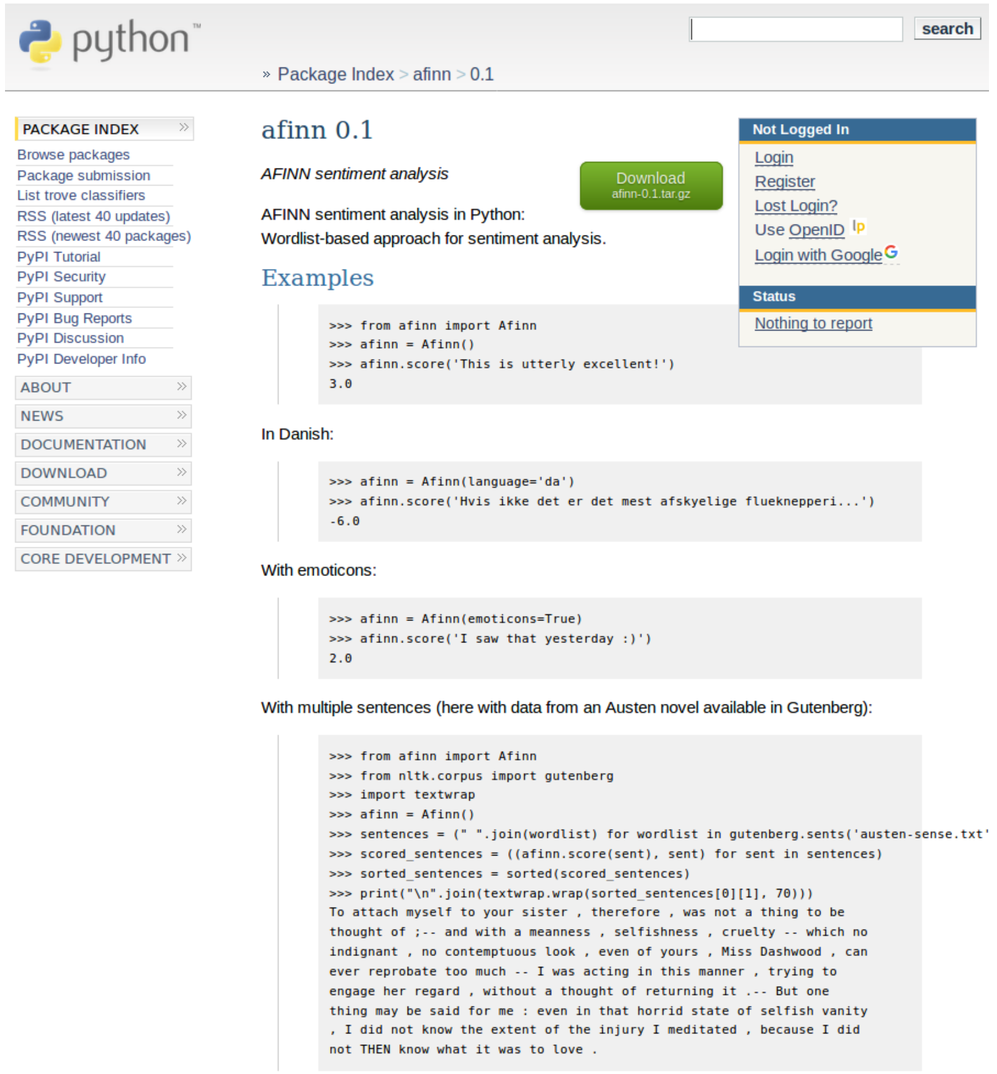
A couple of Jupyter notebooks are available in the GitHub repository.

Used to demonstrate how the module can be applied with a dataset.

GitHub formats the notebook for human readability. It would otherwise be raw JSON.

This notebook computes accuracy on a manually sentiment-scored Twitter dataset.

## Python Package Index



afinn 0.1

*AFINN sentiment analysis*

AFINN sentiment analysis in Python:
Wordlist-based approach for sentiment analysis.

Examples

```
>>> from afinn import Afinn
>>> afinn = Afinn()
>>> afinn.score('This is utterly excellent!')
3.0
```

In Danish:

```
>>> afinn = Afinn(language='da')
>>> afinn.score('Hvis ikke det er det mest afskyelige flueknepperi...')
-6.0
```

With emoticons:

```
>>> afinn = Afinn(emoticons=True)
>>> afinn.score('I saw that yesterday :)')
2.0
```

With multiple sentences (here with data from an Austen novel available in Gutenberg):

```
>>> from afinn import Afinn
>>> from nltk.corpus import gutenberg
>>> import textwrap
>>> afinn = Afinn()
>>> sentences = (" ".join(wordlist) for wordlist in gutenberg.sents('austen-sense.txt'
>>> scored_sentences = ((afinn.score(sent), sent) for sent in sentences)
>>> sorted_sentences = sorted(scored_sentences)
>>> print("\n".join(textwrap.wrap(sorted_sentences[0][1], 70)))
To attach myself to your sister , therefore , was not a thing to be
thought of ;-- and with a meanness , selfishness , cruelty -- which no
indignant , no contemptuous look , even of yours , Miss Dashwood , can
ever reprobate too much -- I was acting in this manner , trying to
engage her regard , without a thought of returning it .-- But one
thing may be said for me : even in that horrid state of selfish vanity
, I did not know the extent of the injury I meditated , because I did
not THEN know what it was to love .
```

afinn distributed from the central open archive *Python Package Index*: https://pypi.python.org/pypi/afinn

Enables others to download the package seamlessly

```
pip install afinn
```

Or search for it with:

```
pip search sentiment
```

Python tools for help with upload.

## Dependencies

Keep dependencies on a bare minimum: None, except standard library (codecs, re, os) — so far.

Otherwise the dependencies should have been added to `requirements.txt`

Example from other package:

```
beautifulsoup4
db.py
docopt
fasttext
flask
Flask-Bootstrap
gensim
jsonpickle
...
```

Enables `pip install -r requirements.txt`

## Issue: Versioneering

Versioneering is a problem at the moment.

Version string “0.1” is hard-coded in the setup file:

```
setup(
    name='afinn',
    packages=['afinn'],
    version='0.1',
...
```

PyPI version is 0.1, but if the GitHub repository is changed this version is no longer reflecting differences.

In the old days, developers would manually update the version.

Now Brain Warner’s `versioneer` can take care of automatically distinguishing git-tagged versions, updated and “dirty” versions.

## Summary

The Python environment has good methods to standardize development.

Python can neatly enforce documentation.

A good number of tools help the developer to write in a best practice mode: testing frameworks, code and documentation style checkers.

Python provides a good framework for publishing open source code.

Persistent and versioned distribution.

Most of the "code" is documentation.

End