afinn project

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

March 28, 2017
afinn project

afinn

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”.

Finn Årup Nielsen
March 28, 2017
GitHub-based development

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):

<table>
<thead>
<tr>
<th>Word</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>abandon</td>
<td>-2</td>
</tr>
<tr>
<td>abandoned</td>
<td>-2</td>
</tr>
<tr>
<td>abandons</td>
<td>-2</td>
</tr>
<tr>
<td>abducted</td>
<td>-2</td>
</tr>
<tr>
<td>abduction</td>
<td>-2</td>
</tr>
<tr>
<td>abductions</td>
<td>-2</td>
</tr>
<tr>
<td>abhor</td>
<td>-3</td>
</tr>
<tr>
<td>abhorred</td>
<td>-3</td>
</tr>
<tr>
<td>abhorrent</td>
<td>-3</td>
</tr>
<tr>
<td>abhors</td>
<td>-3</td>
</tr>
<tr>
<td>abilities</td>
<td>2</td>
</tr>
<tr>
<td>ability</td>
<td>2</td>
</tr>
<tr>
<td>aboard</td>
<td>1</td>
</tr>
<tr>
<td>aborted</td>
<td>-1</td>
</tr>
</tbody>
</table>
Basic Afinn object

The word list is encapsulated 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:

```python
>>> 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:

```python
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.

```python
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.
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.
Documentation 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_.


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 ...

Finn Årup Nielsen 14 March 2017
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 :)

Finn Årup Nielsen

March 28, 2017
Testing with Travis

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

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.
afinn project

**Python Package Index**

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

Finn Årup Nielsen
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:

```python
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