sentidict Documentation

Release 0.0.3

Andy Reagan

Mar 28, 2017

Contents:

CHAPTER 1

Getting Started

TL;DR a simple labMT usage script

Usage

This script uses the language assessment by Mechanical Turk (labMT) word list to score the happiness of a corpus. The labMT word list was created by combining the 5000 words most frequently appearing in four sources: Twitter, the New York Times, Google Books, and music lyrics, and then scoring the words for sentiment on Amazon's Mechanical Turk. The list is described in detail in the publication Dodds' et al. 2011, PLOS ONE, "Temporal Patterns of Happiness and Information in a Global-Scale Social Network: Hedonometrics and Twitter."

Given two corpora, the script "storylab.py" creates a word-shift graph illustrating the words most responsible for the difference in happiness between the two corpora. The corpora should be large (e.g. at least 10,000 words) in order for the difference to be meaningful, as this is a bag-of-words approach. As an example, a random collection of English tweets from both Saturday January 18 2014 and Tuesday January 21 2014 are included in the "example" directory. They can be compared by moving to the test directory, using the command

```
python example.py example-shift.html
```

and opening the file `example-shift.html` in a web browser. For an explanation of the resulting plot, please visit

http://www.hedonometer.org/shifts.html

Installation

Cloning the github directly is recommended, and then installing locally:

```
git clone https://github.com/andyreagan/labMT-simple.git
cd labMT-simple
python setup.py install
```

This repository can also be installed using pip

```
pip install labMTsimple
```

in which case you can download the tests from github and run them, if desired.

Running tests

Tests are based on nose2, `pip install nose2`, and can be run inside the by executing

```
nose2
```

in the root directory of this repository.

This will compare the two days in test/data and print test.html which shifts them, allowing for a changable lens.

Developing with labMT-simple locally

I find it really useful to reload the library when testing it interactively:

```
try:
    reload
except NameError:
    # Python 3
    from importlib import reload
```

Building these docs

Go into the docs directory (activate local `virtualenv` first), and do the following:

```
\rm -rf _build/*
make html
make latexpdf
git add -f *
git commit -am "new docs, probably should just add a pre-commit hook"
```

Note that these docs will build locally in python 2 because the dependencies exist. With python 3 available, these dependencies will be mocked (and this is set for the online readthedocs site).

(`sphinx-apidoc -o . ../labMTsimple` was run once.)

CHAPTER 2

Detailed Examples

Preparing texts

This is simple really: just load the text to be scored into python. I'm using a subset of a couple days of public tweets to text, and I've already put the tweet text into `.txt` files that I load into strings:

```
f = codecs.open("data/18.01.14.txt","r","utf8")
saturday = f.read()
f.close()

f = codecs.open("data/21.01.14.txt","r","utf8")
tuesday = f.read()
f.close()
```

Loading dictionaries

Again this is really simple, just use the `emotionFileReader` function:

```
lang = 'english'
labMT,labMTvector,labMTwordList = emotionFileReader(stopval=0.0,lang=lang,
↪returnVector=True)
```

Then we can score the text and get the word vector at the same time:

```
saturdayValence,saturdayFvec = emotion(saturday,labMT,shift=True,
↪happsList=labMTvector)
tuesdayValence,tuesdayFvec = emotion(tuesday,labMT,shift=True,happsList=labMTvector)
```

But we don't want to use these happiness scores yet, because they included all words (including neutral words). So, set all of the neutral words to 0, and generate the scores:

```
tuesdayStoppedVec = stopper(tuesdayFvec,labMTvector,labMTwordList,stopVal=1.0)
saturdayStoppedVec = stopper(saturdayFvec,labMTvector,labMTwordList,stopVal=1.0)

saturdayValence = emotionV(saturdayStoppedVec,labMTvector)
tuesdayValence = emotionV(tuesdayStoppedVec,labMTvector)
```

CHAPTER 3

Making Wordshifts

I just merged updates to the d3 wordshift plotting into labMTsimple, and combined with phantom crowbar (see previous post), it's easier than ever to use the labMT data set to compare texts.

To make an html page with the shift, you'll just need to have labMT-simple installed. To automate the process into generating svg files, you'll need the phantom crowbar, which depends on phantomjs. To go all the way to pdf, you'll also need inkscape for making vectorized pdfs, or rsvg for making better formatted, but rasterized, versions.

Let's get set up to make shifts automatically. Since they're aren't many dependencies all the way down, start by getting phantomjs installed, then the phantom-crowbar.

Installing phantom-crowbar

For the phantomjs, I prefer to use homebrew:

```
brew update
brew upgrade
brew install phantomjs
```

Then to get the crowbar, clone the git repository.

```
cd ~
git clone https://github.com/andyreagan/phantom-crowbar
```

To use it system-wide, I use the bash alias:

```
alias phantom-crowbar="/usr/local/bin/phantomjs ~/phantom-crowbar/phantom-crowbar.js"
```

Without too much detail, I recommend adding this to your `~/.bash_profile` so that it's loaded every time you start a terminal session.

Installing inkscape

You only need inkscape if you want to go from svg to pdf (and there are other ways too), but this one is easy with, again, homebrew.

```
brew install inkscape
```

Installing rsvg

You only need inkscape if you want to go from svg to pdf (and there are other ways too), but this one is easy with, again, homebrew.

```
brew install librsvg
```

Installing labMTsimple

There are two ways to get it: using pip of cloning the git repo. If you're not sure, use pip. I think pip makes it easier to keep it up to date, etc.

```
pip install labMTsimple
```

Making your first shift

If you cloned the git repository, install the thing and then you can check out the example in `examples/example.py`. If you went with pip, see that file on github.

Go ahead and run that script!

```
python example-002.py
```

You can open the html file to see the shift in any browser, with your choice of local webserver. Python's SimpleHTTPServer works fine, and I've found that the node based http-server is a bit more stable.

To take out the svg, go ahead and use the `phantom-crowbar.js` file copied to the `example/static` directory. Running it looks like this, for me:

```
/usr/local/bin/phantomjs js/shift-crowbar.js example-002.html shiftsvg wordshift.svg
```

Using inkscape or librsvg on my computer look like this:

```
/Applications/Inkscape.app/Contents/Resources/bin/inkscape -f $(pwd)/wordshift.svg -A
↪$(pwd)/wordshift-inkscape.pdf

rsvg-convert --format=eps worshift.svg > wordshift-rsvg.eps
epstopdf wordshift-rsvg.eps
```

And again, feel free to tweet suggestions at @andyreagan, and submit pull requests to the source code!

Full Automation

I've wrapped up all of this into what is potentially the most backwards way to generate figure imaginable. The `shiftPDF()` function operates the same way as the `shiftHTML()`, but uses the headless web server to render the d3 graphic, then exectues a piece of injected JS to save a local SVG, and uses command line image manipulation libraries to massage it into a PDF.

On my macbook, this works, but your mileage will most certainly vary.

CHAPTER 4

Advanced Usage

About Tries

For dictionary lookup of scores from phrases, the fastest benchmarks I've found and that were reasonable stable were from the libraries `datrie` and `marisatrie` which both have python bindings.

They're used in the `speedy` module in an attempt to both speed things up, and match against word stems.

Advanced Parsing

Some dictionaries use word stems to cover the multiple uses of a single word, with a single score. We can very quickly match these word stems using a prefix match on a trie. This is much better than using many compiled RE matches, which in my testing took a very long time.

CHAPTER 5

sentidict package

Submodules

sentidict.dictionaries module

sentidict.utils module

Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search