

Detección de sexismo en Twitter

Ingeniería de Software

Abraham Solís Álvarez
Mario Alejandro Gil Lázaro

December 9, 2016

Contents

Contents	1
1 Requerimientos	2
1.1 Descripción del proyecto	2
1.2 Requerimientos del proyecto	2
1.3 Dependencias	3
2 Documentación	3
2.1 Cómo se usa el software	3
3 Desarrollo	4
3.1 Cronograma de Actividades	4
4 Package analisis-machismo	5
4.1 Modules	5
5 Package analisis-machismo.app	6
5.1 Modules	6
6 Module analisis-machismo.app.analysis	7
6.1 Functions	7
7 Module analisis-machismo.app.dictionary_tagger	8
7.1 Class DictionaryTagger	8
7.1.1 Methods	8
8 Module analisis-machismo.app.key_reader	9
8.1 Class KeyReader	9
8.1.1 Methods	9
9 Module analisis-machismo.app.tag_counter	10
9.1 Class TagCounter	10
9.1.1 Methods	10

10 Module analisis-machismo.app.tweet_formatter	11
10.1 Class TweetFormatter	11
10.1.1 Methods	11
11 Module analisis-machismo.app.twitter_miner	12
11.1 Class TwitterMiner	12
11.1.1 Methods	12
12 Package analisis-machismo.tests	13
12.1 Modules	13
13 Module analisis-machismo.tests.test_dictionary_tagger	14
13.1 Class TestDictionaryTagger	14
13.1.1 Methods	14
14 Module analisis-machismo.tests.test_key_reader	15
14.1 Class TestKeyReader	15
14.1.1 Methods	15
15 Module analisis-machismo.tests.test_tweet_formatter	16
15.1 Class TestTweetFormatter	16
15.1.1 Methods	16
Index	17

1 Requerimientos

1.1 Descripción del proyecto

Este proyecto busca determinar los niveles de sexismo y otras manifestaciones de odio y discriminación, en el texto que los usuarios del sitio de microblogueo Twitter, escriben diariamente. Dado que el internet es una plataforma moderna de expresión y debido también a que la ya mencionada red social posee un número importante de usuarios, consideramos que la información ahí recabada representa una muestra importante. Así mismo, el hecho de que los medios virtuales son comúnmente considerados como medios poco trascendentales, en los que se puede supuestamente evitar las consecuencias que los propios comentarios puedan ocasionar, creemos que las opiniones ahí expresadas poseen un alto grado de sinceridad, mayor al que podría obtenerse en el discurso hablado.

1.2 Requerimientos del proyecto

El sistema propuesto debe ser capaz de recolectar los posts directamente del stream de Twitter, convertir los datos al formato que mejor convenga, etiquetar palabra por palabra el texto, y realizar mediciones que permitan elaborar indicadores que arrojen luz sobre las costumbres y la cultura de los usuarios de la red social. Se pretende lograr esto manteniendo los más altos estándares de calidad que sea posible, aplicando prácticas de programación modernas.

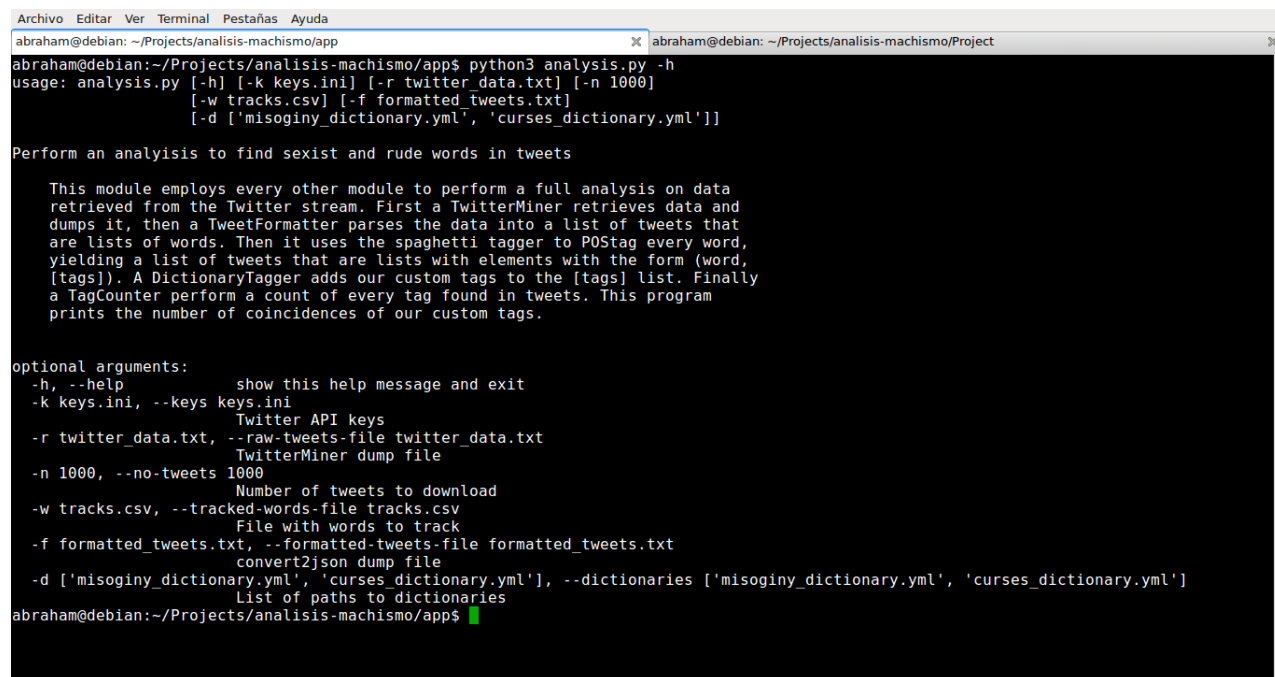
1.3 Dependencias

Para ejecutar el programa con las mayores probabilidades de éxito se recomienda utilizar un sistema operativo basado en GNU/Linux. Es necesario instalar una Python 3.x, de preferencia la versión 3.5, que fue la utilizada en el desarrollo del sistema. Se recomienda utilizar el programa ‘pip’ para instalar los módulos de Python ‘nltk’ y ‘plac’, que son necesarios para ejecutar el programa. Es necesario contar con una conexión a internet.

2 Documentación

2.1 Cómo se usa el software

Por el momento, la única forma de interactuar con el sistema es mediante el módulo ‘analysis.py’, localizado en el directorio ‘app’ (con el comando ‘python analysis.py’). Tenga en cuenta que usted debe estar localizado dentro de dicho directorio. Además, se proporciona un script que ejecuta las pruebas del sistema en el directorio raíz: ‘run_tests’.



```
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
abraham@debian: ~/Projects/analisis-machismo/app
abraham@debian: ~/Projects/analisis-machismo/Project
abraham@debian:~/Projects/analisis-machismo/app$ python3 analysis.py -h
usage: analysis.py [-h] [-k keys.ini] [-r twitter_data.txt] [-n 1000]
                  [-w tracks.csv] [-f formatted_tweets.txt]
                  [-d ['misoginy_dictionary.yml', 'curses_dictionary.yml']]

Perform an analysis to find sexist and rude words in tweets

This module employs every other module to perform a full analysis on data
retrieved from the Twitter stream. First a TwitterMiner retrieves data and
dumps it, then a TweetFormatter parses the data into a list of tweets that
are lists of words. Then it uses the spaghetti tagger to POSTag every word,
yielding a list of tweets that are lists with elements with the form (word,
[tags]). A DictionaryTagger adds our custom tags to the [tags] list. Finally
a TagCounter perform a count of every tag found in tweets. This program
prints the number of coincidences of our custom tags.

optional arguments:
  -h, --help            show this help message and exit
  -k keys.ini, --keys keys.ini
                        Twitter API keys
  -r twitter_data.txt, --raw-tweets-file twitter_data.txt
                        TwitterMiner dump file
  -n 1000, --no-tweets 1000
                        Number of tweets to download
  -w tracks.csv, --tracked-words-file tracks.csv
                        File with words to track
  -f formatted_tweets.txt, --formatted-tweets-file formatted_tweets.txt
                        convert2json dump file
  -d ['misoginy_dictionary.yml', 'curses_dictionary.yml'], --dictionaries ['misoginy_dictionary.yml', 'curses_dictionary.yml']
                        List of paths to dictionaries
abraham@debian:~/Projects/analisis-machismo/app$
```

3 Desarrollo

3.1 Cronograma de Actividades

Cronograma de actividades

De acuerdo con las etapas que se deben llevar a cabo para la realización del proyecto, se presenta el siguiente cronograma de commits.

Commit \ Días	Sat Dec 3	Sun Dec 4	Mon Dec 5	Tue Dec 6	Wed Dec 7	Sat Dec 8
Initial commit						
Read keys from .ini file						
Add machist dictionary						
Add readkeys unit tests						
Separate app and tests code						
Data belongs to app folder						
Write script to automate tests						
Read tracks file						
Add filter tracks file						
Update local repository						
Change to read actual track file						
Remove quotes from tracks file						
Update visualizer to show only the text of the tweet						
Add KeyboardInterrupted Exception						
Add filter tracks file						
Update machist dictionary						
Sexist tagger ready to be plugged						
Update local repository						
Update .gitignore file						
Bring spaghetti files to use trained taggers						
Resolve .gitignore conflict						
Make key_reader object oriented						
Use more concise names						
Pack spaghetti module						
Refactor twitter_streaming module to make TwitterMiner class						
Reduce console output when keyboard interrupt						
Refactor visualizer.py code to make it OO						
Add regex to clean tweets in tweet_formatter						
Add tests for tweet_formatter.py and make it pass						
Test compile_dictionaries from dictionary_tagger						
Test tag method in dictionary_tagger						
Add 'stop after # tweets retrieved' functionality to listener						
Write some docstrings						
Add some docstring						
Split dictionaries						
Changed dictionaries						
Repaired tabs inconsistencies						
Add some docstring						
Use plac to generate -h menu for analysis.py						
Update local repository						
Add some docstring						
Add a tag counter						
Update local repository						
Add a usage prompt to main function with plac						
Improve analysis documentation						
Review key_reader documentation						
Rewrite tag_counter documentation						
Update tweet_formatter documentation						
Update twitter_miner documentation						

4 Package analisis-machismo

4.1 Modules

- **app** (*Section 5, p. 6*)
 - **analysis** (*Section 6, p. 7*)
 - **dictionary_tagger** (*Section 7, p. 8*)
 - **key_reader** (*Section 8, p. 9*)
 - **tag_counter** (*Section 9, p. 10*)
 - **tweet_formatter** (*Section 10, p. 11*)
 - **twitter_miner** (*Section 11, p. 12*)
- **tests** (*Section 12, p. 13*)
 - **test_dictionary_tagger** (*Section 13, p. 14*)
 - **test_key_reader** (*Section 14, p. 15*)
 - **test_tweet_formatter** (*Section 15, p. 16*)

5 Package analisis-machismo.app

5.1 Modules

- **analysis** (*Section 6, p. 7*)
- **dictionary_tagger** (*Section 7, p. 8*)
- **key_reader** (*Section 8, p. 9*)
- **tag_counter** (*Section 9, p. 10*)
- **tweet_formatter** (*Section 10, p. 11*)
- **twitter_miner** (*Section 11, p. 12*)

6 Module analisis-machismo.app.analysis

6.1 Functions

```
main(keys='keys.ini', raw_tweets_file='twitter_data.txt', no_tweets=1000,  
tracked_words_file='tracks.csv', formatted_tweets_file='formatted_tweets.txt',  
dictionaries=['misoginy_dictionary.yml', 'curses_dictionary.yml'])
```

Perform an analysis to find sexist and rude words in tweets

This module employs every other module to perform a full analysis on data retrieved from the Twitter stream. First a TwitterMiner retrieves data and dumps it, then a TweetFormatter parses the data into a list of tweets that are lists of words. Then it uses the spaghetti tagger to POSTag every word, yielding a list of tweets that are lists with elements with the form (word, [tags]). A DictionaryTagger adds our custom tags to the [tags] list. Finally a TagCounter perform a count of every tag found in tweets. This program prints the number of coincidences of our custom tags.

7 Module `analisiis-machismo.app.dictionary_tagger`

7.1 Class `DictionaryTagger`

Python class for tagging text with dictionaries

7.1.1 Methods

<code>__init__</code> (<i>self</i> , <i>dictionary_paths</i>)
--

Dictionary is a dict containing all the dictionaries parsed from the paths given.

<code>compile_dictionaries</code> (<i>self</i> , <i>dictionary_paths</i>)
--

Returns a list of dictionaries parsed from .yaml files
--

<code>tag</code> (<i>self</i> , <i>postagged_sentences</i>)
--

<code>tag_sentence</code> (<i>self</i> , <i>sentence</i> , <i>tag_with_lemmas</i> =None)
--

The result is only tagging of all the possible ones. The resulting tagging is determined by these two priority rules:

- | |
|---|
| <ul style="list-style-type: none">• longest matches have higher priority• search is made left to right |
|---|

8 Module *analysis-machismo.app.key_reader*

8.1 Class `KeyReader`

Wrapper of `ConfigParser` to load `.ini` file with Twitter API keys

8.1.1 Methods

<code>__init__(self)</code>

<code>read(self, filename='keys.ini', section='keys')</code>
--

Import the configparser, tell it to read the file, and get a listing of the sections. Sections are listed in a python dictionary.

9 Module *analysis-machismo.app.tag_counter*

9.1 Class `TagCounter`

Wrapper for a tag counting method

9.1.1 Methods

<code>__init__</code> (<i>self</i> , <i>tagged_tweets</i>)
Store a list of tweets in case none is provided in <code>count()</code>

<code>count</code> (<i>self</i> , <i>tagged_tweets</i> =None)
Handle the counting of tags inserting them in a dictionary to ensure their uniqueness. Can manage a list of tags and a single string tag.

10 Module *alisis-machismo.app.tweet_formatter*

10.1 Class `TweetFormatter`

Provides the tools needed to format raw data from Twitter stream to stripped text like this: raw -> json -> text -> stripped text.

10.1.1 Methods

<code>__init__</code> (<i>self</i> , <i>source_file</i> =None)
--

<code>convert2json</code> (<i>self</i> , <i>tweets_source</i> =None)
--

Wrap json module to convert raw Twitter data to json
--

<code>convert2text</code> (<i>self</i> , <i>tweets_data</i> =None, <i>output_file</i> =None)
--

Grab 'text' field of jsons only and return a list of them

<code>clean_tweets</code> (<i>self</i> , <i>tweets_text</i> =None)
--

Regular expressions to remove unnecessary characters in Tweets
--

11 Module *analisiis-machismo.app.twitter_miner*

11.1 Class `TwitterMiner`

Exposes the whole chain needed to retrieve data from the Twitter stream from reading the keys, create an oAuth object,

11.1.1 Methods

<code>__init__</code> (<i>self</i> , <i>keys_file</i> ='keys.ini', <i>output_file</i> =None, <i>max_tweets</i> =10)

<code>connect</code> (<i>self</i>)

Get keys and connect to Twitter through OAuth

<code>mine</code> (<i>self</i> , <i>track_words</i> , <i>output_file</i> =None)

Retrieve tweets with text that matches track-words.

12 Package analisis-machismo.tests

12.1 Modules

- `test_dictionary_tagger` (*Section 13, p. 14*)
- `test_key_reader` (*Section 14, p. 15*)
- `test_tweet_formatter` (*Section 15, p. 16*)

13 Module `analysis-machismo.tests.test_dictionary_tagger`

13.1 Class `TestDictionaryTagger`

`unittest.TestCase` — `analysis-machismo.tests.test_dictionary_tagger.TestDictionaryTagger`

13.1.1 Methods

<code>test_init_output(self)</code>
<code>unittest</code> supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. The <code>unittest</code> module provides classes that make it easy to support these qualities for a set of tests.
<code>test_tag_sentences(self)</code>

14 Module `analysis-machismo.tests.test_key_reader`

14.1 Class `TestKeyReader`

`unittest.TestCase` —
`analysis-machismo.tests.test_key_reader.TestKeyReader`

14.1.1 Methods

<code>setup(self)</code>

<code>test_missing_section(self)</code>
--

unittest supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. The unittest module provides classes that make it easy to support these qualities for a set of tests.

<code>test_valid_files(self)</code>
--

15 Module `analysis-machismo.tests.test_tweet_formatter`

15.1 Class `TestTweetFormatter`

`unittest.TestCase` — `analysis-machismo.tests.test_tweet_formatter.TestTweetFormatter`

The crux of each test is a call to `assertEqual()` to check for an expected result; `assertTrue()` or `assertFalse()` to verify a condition; or `assertRaises()` to verify that a specific exception gets raised. These methods are used instead of the `assert` statement so the test runner can accumulate all test results and produce a report.

15.1.1 Methods

test_inexistent_file(*self*)

test_no_JSON_tweets(*self*)

test_no_text_tweets(*self*)

test_no_text_key(*self*)

Index

- analysis-machismo (*package*), 2
 - analysis-machismo.app (*package*), 3
 - analysis-machismo.app.analysis (*module*), 4
 - analysis-machismo.app.dictionary_tagger (*module*), 5
 - analysis-machismo.app.key_reader (*module*), 6
 - analysis-machismo.app.tag_counter (*module*), 7
 - analysis-machismo.app.tweet_formatter (*module*), 8
 - analysis-machismo.app.twitter_miner (*module*), 9
 - analysis-machismo.tests (*package*), 10
 - analysis-machismo.tests.test_dictionary_tagger (*module*), 11
 - analysis-machismo.tests.test_key_reader (*module*), 12
 - analysis-machismo.tests.test_tweet_formatter (*module*), 13