Detección de sexismo en Twitter
Ingeniería de Software

Abraham Solís Álvarez
Mario Alejandro Gil Lázaro

December 9, 2016

Contents

1 Descripción

Este proyecto busca determinar los niveles de sexismo y otras manifestaciones de odio y discriminación, en el texto que los usuarios del sitio de microblogeo 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.

2 Requerimientos

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.

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.

4 Modo de uso

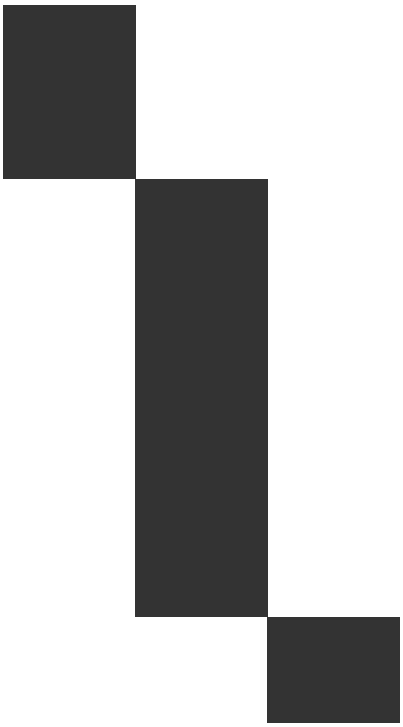
Por el momento, la única forma de interactuar con el sistema es mediante el módulo 'analysis.py', localizado en el directorio 'app' (con el comando 'python analysis.py'). Tenga en cuenta que usted debe estar localizado dentro de dicho directorio. Además, se proporciona un script que ejecuta las pruebas del sistema en el directorio raíz: '$\text{run}_t ests$'.

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 KeyboadInterrupted 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



5 Package analisis-machismo

5.1 Modules

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

6 Package analisis-machismo.app

6.1 Modules

7 Module analisis-machismo.app.analysis

7.1 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 analyisis 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.

8 Module analisis-machismo.app.dictionary_tagger

8.1 Class DictionaryTagger

Python class for tagging text with dictionaries

8.1.1 Methods

__init__(*self*, *dictionary_paths*)

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

compile_dictionaries(*self*, *dictionary_paths*)

Returns a list of dictionaries parsed from .yml files

tag(*self*, *postagged_sentences*)

tag_sentence(*self*, *sentence*, *tag_with_lemmas*=`None`)

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

- longest matches have higher priority
- search is made left to right

9 Module analisis-machismo.app.key_reader

9.1 Class KeyReader

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

9.1.1 Methods

__**init**__(*self*)

read(*self*, *filename*=`'keys.ini'`, *section*=`'keys'`)

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

10 Module analisis-machismo.app.tag_counter

10.1 Class TagCounter

Wrapper for a tag counting method

10.1.1 Methods

__init__(*self*, *tagged_tweets*)

Store a list of tweets in case none is provided in count()

count(*self*, *tagged_tweets*=`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.

11 Module analisis-machismo.app.tweet_formatter

11.1 Class TweetFormatter

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

11.1.1 Methods

__init__(*self*, *source_file*=`None`)

convert2json(*self*, *tweets_source*=`None`)

Wrap json module to convert raw Twitter data to json

convert2text(*self*, *tweets_data*=`None`, *output_file*=`None`)

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

clean_tweets(*self*, *tweets_text*=`None`)

Regular expressions to remove unnecessary characters in Tweets

12 Module analisis-machismo.app.twitter_miner

12.1 Class TwitterMiner

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

12.1.1 Methods

__init__(*self*, *keys_file*='keys.ini', *output_file*=None, *max_tweets*=10)

connect(*self*)

Get keys and connect to Twitter through OAuth

mine(*self*, *track_words*, *output_file*=None)

Retrieve tweets with text that matches track_words.

13 Package analisis-machismo.tests

13.1 Modules

14 Module analisis-machismo.tests.test_dictionary_tagger

14.1 Class TestDictionaryTagger

unittest.TestCase
└ **analisis-machismo.tests.test_dictionary_tagger.TestDictionaryTagger**

14.1.1 Methods

test_init_output(*self*)

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.

test_tag_sentences(*self*)

15 Module analisis-machismo.tests.test_key_reader

15.1 Class TestKeyReader

unittest.TestCase
└ **analisis-machismo.tests.test_key_reader.TestKeyReader**

15.1.1 Methods

setup(*self*)

test_missing_section(*self*)

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.

test_valid_files(*self*)

16 Module analisis-machismo.tests.test_tweet_formatter

16.1 Class TestTweetFormatter

unittest.TestCase
└ **analisis-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.

16.1.1 Methods

test_inexistent_file(*self*)

test_no_JSON_tweets(*self*)

test_no_text_tweets(*self*)

test_no_text_key(*self*)

Index