Notebook-PLA2

July 16, 2019

M2.856 ů Análisis de sentimientos y redes sociales
Máster universitario de Ciencia de datos (Data Science)
Estudios de Informática, Multimedia y Telecomunicación

1 Módulo 2: Extracción de sentimientos y opiniones

1.1 Scripts de los métodos explicados en el módulo

A continuación, se presentan los scripts que ejemplifican los métodos explicados en PLA-2.

Paquetes requeridos para ejecutar este notebook, además de los comentados en el notebook anterior (Notebook-PLA1): - rdflib y neuralcoref (previa instalación de spaCy). Para la instalación podéis usar "pip" o "conda", según el sistema que tengáis instalado.

- Recordad que tenéisq que mantener la misma estructura de las carpetas para que el código encuentre los ficheros .csv y las imágenes que se usan.

Nota: Se recomienda emplear la versión 3.6 de Python.

1.2 1. Detección de los componentes de una opinión

1.2.1 1.1 Detección de las distintas denominaciones del target

Mostramos cómo se detecta el target en sus distintas variaciones:
Hiperónimo
Sinónimo
Referencia anafórica

Comprobar si el término A es un hiperónimo del término B consultando DBpedia

```
In [31]: #Importamos la librería para poder hacer las consultas en la DBpedia
         #La respuesta de DBpedia es un grafo y la información se estructura en formato RDF
         import rdflib
         from rdflib import Graph, URIRef, RDF

         #Método para saber si A es un hiperónimo de B
         def is_hyperonym(source,target):
             dbpedia_uri = URIRef('http://dbpedia.org/resource/'+ source) #Uri que apunta a A
```

```
            g = Graph()
            g.parse(dbpedia_uri)#Se parsea la respuesta de DBpedia en forma de grafo
            url_matches = []
            for obj in g.objects(subject=dbpedia_uri, predicate=RDF.type): #Buscamos en el gr
                #sobre los objetos del predicado 'type' cuando el sujeto es la uri que apunta
                #Se recogen las referencias al objeto de 'type' según Yago en las que esté B
                if 'http://dbpedia.org/class/yago/' in obj and target in obj:
                    url_matches.append(obj)
            #Si hay referencias al objeto de type que contienen B, entonces A es su hiperónim
            if len(url_matches) > 0:
                print(target, "ES UN HIPERÓNIMO DE", source)
            else:
                print(target, "NO ES UN HIPERÓNIMO DE ", source)

        is_hyperonym('IPhone','Telephone')
```

```
Telephone ES UN HIPERÓNIMO DE IPhone
```

Comprobar si A es un sinónimo de B con word embeddings.

```
In [32]: import gensim
         import nltk
         from nltk.corpus import wordnet as wn

         #Cargamos el modelo Word2Vec de opiniones sobre smartphones
         model = gensim.models.Word2Vec.load('MODEL-MOBILEPHONE-REVIEWS-PL2')

         term = 'samsung'

         #Obtenemos los phrases más similares al término a partir del modelo Word2Vec
         most_similars = model.wv.most_similar(positive=term)

         print("PHRASES CON VALOR DE SIMILITUD", most_similars)

         #De la lista de phrases más similares (score de similitud > 0.4) tomamos los sintagma
         #(NN, NS o JJ+NN, NN+NN, NN+NS)

         def is_np(candidate):
             test = False
             tokens = candidate.split()
             tagged_tokens = nltk.pos_tag(tokens)
             #Si es un término multipalabra
             if len(tagged_tokens) > 1:
                 PoS_initial = tagged_tokens[0][1][:2]
                 PoS_final = tagged_tokens[-1][1][:2]
                 if ((PoS_initial == 'NN' or PoS_initial == 'JJ') and (PoS_final == 'NN' or PoS
                     test = True
```

```
            #Si es un término monopalabra
            else:
                if len(candidate) > 1 and tagged_tokens[0][1][:2] == 'NN' or tagged_tokens[0]
                    test = True
            return test


        target_candidates = [tc[0] for tc in most_similars if tc[1] > 0.4 and is_np(tc[0]) ==

        print("CANDIDATOS A TARGET", target_candidates)
```

```
PHRASES CON VALOR DE SIMILITUD [('samsung products', 0.5297492146492004), ('htc', 0.5151442885
CANDIDATOS A TARGET ['samsung products', 'htc', 'lg', 'galaxy', 'touch wiz', 'touchwiz', 'samsu
```

Referencia anafórica al target

```
In [33]: import neuralcoref
         import spacy
         import pytorch

         nlp = spacy.load('en_coref_sm')
         #o import en_coref_sm nlp = en_coref_sm.load()
         #o en_coref_lg para un modelo entrenado con un corpus más grande

         doc = nlp(u'I bought an iPhone a few days ago. It was such a nice phone. \
         The touch screen was really cool. The voice quality was clear too. \
         However, my mother was mad with me as I did not tell her before I bought it. \
         Although the battery life was not long, that is ok for me')

         doc._.coref_resolved


        ---------------------------------------------------------------------------

        ModuleNotFoundError                       Traceback (most recent call last)

        <ipython-input-33-1aed568fae11> in <module>
          1 import neuralcoref
          2 import spacy
    ----> 3 import pytorch
          4
          5 nlp = spacy.load('en_coref_sm')


        ModuleNotFoundError: No module named 'pytorch'
```

1.2.2 1.2 Detección de los aspectos

Obtener las propiedades de un producto según una infobox de la Wikipedia

In [34]:
```
import requests

#Obtener las propiedades del target 'IPhone', consultando el grafo de DBpedia
#y obteniendo la información en formato RDF

def get_properties(target):
    data = requests.get("http://dbpedia.org/data/" + target + ".json").json()
    uris = data["http://dbpedia.org/resource/" + target]
    properties = []

    for u in uris:
        if '/property/' in u:
            us = u.split('/')
            properties.append(us[-1])
    return properties

target = 'IPhone'

properties_target = get_properties(target)

print(properties_target)
```

```
['unitssold', 'developer', 'reason', 'sound', 'date', 'dimensions', 'power', 'caption', 'displa
```

Detección de los aspectos por clustering de sus synsets de Wordnet

In [35]:
```
#De una opinión, tomamos los candidatos a ser aspectos del target

import nltk
from nltk import word_tokenize
from nltk.util import ngrams
from nltk.collocations import *
import re
import collections

#Un candidato a ser aspecto es un término monopalabra o multipalabra que forma un sin

def is_np(candidate):
    test = False
    tokens = candidate.split()
    tagged_tokens = nltk.pos_tag(tokens)
    if len(tagged_tokens) > 1:
        PoS_initial = tagged_tokens[0][1][:2]
        PoS_final = tagged_tokens[-1][1][:2]
```

```
            if ((PoS_initial == 'NN' or PoS_initial == 'JJ') and (PoS_final == 'NN' or PoS
                test = True
        else:
            if len(candidate) > 1 and tagged_tokens[0][1][:2] == 'NN' or tagged_tokens[0]
                test = True
        return test

    def get_np_candidates(text):
        #Definimos las métricas que evaluaran si un n-grama puede ser una collocation
        bigram_measures = nltk.collocations.BigramAssocMeasures()
        #Tokenizamos y obtenemos los tokens del texto
        tokens = [w for w in word_tokenize(text.lower())]
        #Búsqueda de bigramas
        bigramfinder = BigramCollocationFinder.from_words(tokens)
        #Se toman los bigramas que no tienen signos de puntuación, etc.
        bigramfinder.apply_word_filter(lambda w: (re.match(r'\W', w)))
        #N mejores candidatos a ser colocaciones, tras pasar por el cálculo del PMI
        bigram_candidates = bigramfinder.nbest(bigram_measures.pmi,100)
        #Transformación de la tupla del bigrama a collocation
        collocation_candidates = [" ".join(bc) for bc in bigram_candidates]
        #Elegimos los candidatos que son sintagmas nominales
        np_candidates = [c for c in tokens + collocation_candidates if is_np(c) == True]
        return np_candidates

    opinion = "I bought an iPhone a few days ago. It was such a nice phone. The touch scre
    The voice quality was clear too. Although the battery life was not long, that is ok fo
    my mother was mad with me as I did not tell her before I bought it. She also thought t
    and wanted me to return it to the shop"

    np_candidates = get_np_candidates(opinion)
```

In [36]:
```
#Transformamos los candidatos a aspecto a sus synsets de Wordnet, cuando estos synset

from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()

np_candidates.append('telephone') # Ponemos el hiperónimo de iPhone

candidates_synsets_list = []

for npc in np_candidates:
    candidates_synsets = wn.synsets(lem.lemmatize(npc.replace(' ','_')),pos=wn.NOUN)
    for cs in candidates_synsets:
        if "'" + npc.replace(' ','_') + ".n" in str(cs):
            candidates_synsets_list.append(cs)
```

In [37]:
```
#Transformamos las propiedades del target, según Wikipedia, a sus synsets de Wordnet,
```

```
        import requests

        #Obtener las propiedades del target 'IPhone', según Wikipedia

        def get_properties(target):
            data = requests.get("http://dbpedia.org/data/" + target + ".json").json()
            uris = data["http://dbpedia.org/resource/" + target]
            properties = []

            for u in uris:
                if '/property/' in u:
                    us = u.split('/')
                    properties.append(us[-1])
            return properties

        target = 'IPhone'

        properties_target = get_properties(target)

        features_synsets_list = []

        for pt in properties_target:
            features_synsets = wn.synsets(lem.lemmatize(pt.replace(' ','_')),pos=wn.NOUN)
            for fs in features_synsets:
                if "'" + pt + ".n" in str(fs):
                    features_synsets_list.append(fs)
```

```
In [38]: #Creamos los vectores que representan a los candidatos a aspecto
         #Los vectores se construyen calculando la distancia de cada synset del candidato a as
         #la propiedad del target

         from numpy import matrix

         def create_vector(synset, synsets_vocabulary):
             vector = [s.wup_similarity(synset) for s in synsets_vocabulary] #Wu and Palmer sc
             return vector

         vectors = [create_vector(v,features_synsets_list) for v in candidates_synsets_list]

         X = matrix(vectors)

         print(X)
```

```
[[0.16666667 0.13333333 0.33333333 ... 0.30769231 0.23529412 0.26666667]
 [0.15384615 0.125      0.46153846 ... 0.57142857 0.44444444 0.5       ]
 [0.125      0.10526316 0.375      ... 0.35294118 0.28571429 0.31578947]
 ...
```

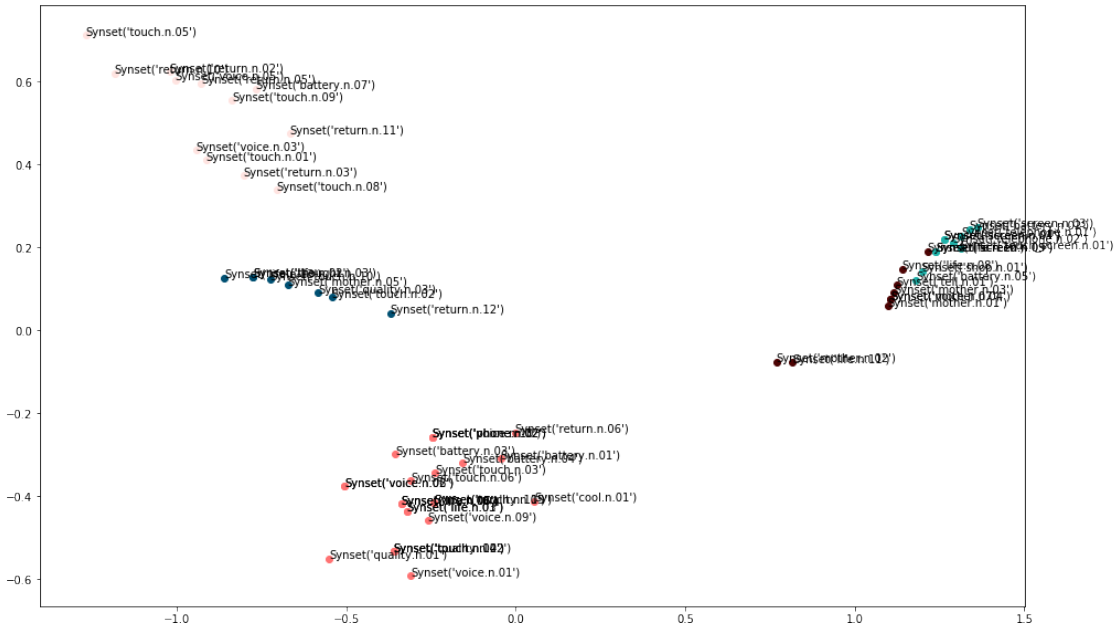
```
 [0.38095238 0.57142857 0.11111111 ... 0.10526316 0.08695652 0.0952381 ]
 [0.44444444 0.77777778 0.13333333 ... 0.125      0.1        0.11111111]
 [0.44444444 0.66666667 0.13333333 ... 0.125      0.1        0.11111111]]
```

In [39]:
```python
from sklearn.cluster import KMeans

#Clustering de los vectores

num_clusters = 5

km = KMeans(n_clusters=num_clusters, n_init=10) # n_init para mantener la consistenci

km.fit(X)

labels = km.labels_.tolist()

labels_color_map = {
    0: '#20b2aa', 1: '#ff7373', 2: '#ffe4e1', 3: '#005073', 4: '#4d0404'
}

class_0 = []
class_1 = []
class_2 = []
class_3 = []
class_4 = []

for i in range(len(labels)):
    if labels[i] == 0:
        class_0.append(candidates_synsets_list[i])
    if labels[i] == 1:
        class_1.append(candidates_synsets_list[i])
    if labels[i] == 2:
        class_2.append(candidates_synsets_list[i])
    if labels[i] == 3:
        class_3.append(candidates_synsets_list[i])
    if labels[i] == 4:
        class_4.append(candidates_synsets_list[i])

print("CLASS_0", class_0)
print("CLASS_1", class_1)
print("CLASS_2", class_2)
print("CLASS_3", class_3)
print("CLASS_4", class_4)
```

```
CLASS_0 [Synset('screen.n.01'), Synset('screen.n.03'), Synset('screen.n.04'), Synset('screen.n
CLASS_1 [Synset('phone.n.02'), Synset('touch.n.03'), Synset('touch.n.04'), Synset('touch.n.06')
CLASS_2 [Synset('touch.n.01'), Synset('touch.n.05'), Synset('touch.n.08'), Synset('touch.n.09')
```

```
CLASS_3 [Synset('touch.n.02'), Synset('touch.n.10'), Synset('quality.n.03'), Synset('life.n.02
CLASS_4 [Synset('voice.n.07'), Synset('life.n.08'), Synset('life.n.10'), Synset('life.n.11'), S
```

```
In [49]: #Visualización

         from sklearn.decomposition import PCA
         import matplotlib.pyplot as plt

         reduced_data = PCA(n_components=2).fit_transform(X)

         fig, ax = plt.subplots(figsize=(16,10))
         for index, instance in enumerate(reduced_data):
             pca_comp_1, pca_comp_2 = reduced_data[index]
             color = labels_color_map[labels[index]]
             ax.scatter(pca_comp_1, pca_comp_2, c=color)
             ax.annotate(s=str(candidates_synsets_list[index]),xy=(pca_comp_1, pca_comp_2))
         plt.show()
```



Detección de los aspectos a partir de opinion words

```
In [50]: #Obtenemos los candidatos a ser aspectos y la lista de opinion words

         import gensim
         from nltk.stem.wordnet import WordNetLemmatizer

         lem = WordNetLemmatizer()
```

```
        #Los candidatos a aspecto ya han sido obtenidos antes (np_candidates).
        #Solo falta lematizarlos

        print(np_candidates)

        #Creamos una lista de opinion words a partir del diccionario AFINN
        opinion_words_file = 'AFINN-111.txt'

        opinion_words = []

        with open(opinion_words_file, 'r') as of:
            ol = of.readlines()
            for wo in ol:
                w, o = wo.strip().split('\t')
                opinion_words.append(w)

        #Obtenemos los lemas unificados de los candidatos a aspecto si no están en la lista d

        def unify(l):
            term_unified = l
            if l == 'touch_screen':
                term_unified = 'touchscreen'
            return term_unified

        aspect_candidates = [unify(lem.lemmatize(ac.replace(' ','_'))) for ac in np_candidates

        print(aspect_candidates)
```

```
['bought', 'iphone', 'days', 'phone', 'touch', 'screen', 'cool', 'voice', 'quality', 'battery'
['bought', 'iphone', 'day', 'phone', 'touch', 'screen', 'voice', 'quality', 'battery', 'life',
```

In [51]:
```
        #Cargamos el modelo de opiniones sobre smartphones

        model= gensim.models.Word2Vec.load('MODEL-MOBILEPHONE-REVIEWS-PL2')

        #Creamos el vocabulario, que es la lista de candidatos a término y las opinion words
        #y en el modelo

        opinion_words_in_opinion = [ow for ow in word_tokenize(opinion.lower()) if ow in opini

        vocabulary = aspect_candidates + opinion_words_in_opinion

        vocabulary_in_model = [v.replace('_',' ') for v in vocabulary if v.replace('_',' ') in

        #Creamos los vectores de los elementos del vocabulario, calculando la distancia (cosi
        #vector del término del vocabulario según el modelo de opiniones de smartphones y los
```

```
        #elementos del vocabulario según el mismo modelo

        import numpy as np

        def sem_distance(a, o):
            cosine_similarity = np.dot(model[a], model[o]) / (np.linalg.norm(model[a])* np.li
            return cosine_similarity


        def create_vector(v, vocabulary):
            vector = [sem_distance(v, vb) for vb in vocabulary]
            return vector

        vectors = [create_vector(v,vocabulary_in_model) for v in vocabulary_in_model]
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launch
```

```
In [52]: #Clustering

        from numpy import matrix
        from sklearn.cluster import KMeans
        from sklearn.decomposition import PCA
        import matplotlib.pyplot as plt

        X = matrix(vectors)

        num_clusters = 3

        km = KMeans(n_clusters=num_clusters, n_init=5) # initial number in order to keep cons

        km.fit(X)

        labels = km.labels_.tolist()

        class_0 = []
        class_1 = []
        class_2 = []

        for i in range(len(labels)):
            if labels[i] == 0:
                class_0.append(vocabulary_in_model[i])
            if labels[i] == 1:
                class_1.append(vocabulary_in_model[i])
            if labels[i] == 2:
                class_2.append(vocabulary_in_model[i])

        print("CLASS_0", class_0)
```

```
        print("CLASS_1", class_1)
        print("CLASS_2", class_2)
```

```
CLASS_0 ['iphone', 'touch', 'screen', 'voice', 'quality', 'ok', 'tell', 'touchscreen', 'voice c
CLASS_1 ['bought', 'mother', 'bought', 'thought', 'return', 'shop', 'telephone', 'mad']
CLASS_2 ['day', 'phone', 'battery', 'life', 'phone', 'battery life']
```

1.2.3 1.3 Detección de la polaridad

Preprocesado

In [53]:
```
#Preprocesado de twits

import re

##Quitar emojis

text = "Adicto a la lectura. Apasionado de la política y la economía. Balonmano, natu
       Dibujo tebeos como croquetas, feministo  y... a veces soy Clara te canta "

emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags=re.UNICODE)
print(text)
print(emoji_pattern.sub(r'', text)) # no emoji

## Quitar urls

text = "AEP es un proyecto para difundir material anarquista existente. https://t.co/

no_http_text = re.sub(r'(\s)http\S+', '', text) #\S+ matches any non-whitespace chara

print (text)
print(no_http_text)

## Quitar hashtags

text = "Tinc 4 #PremisTuitort per ser cuqui i fer #Rubiadas, dos @GatsMalvats"

print(text)

no_hash_arr_text = text = re.sub(r'[#@]', '', text)

print (no_hash_arr_text)
```

```
Adicto a la lectura. Apasionado de la política y la economía. Balonmano, naturaleza y mucha mús
Adicto a la lectura. Apasionado de la política y la economía. Balonmano, naturaleza y mucha mús
AEP es un proyecto para difundir material anarquista existente. https://t.co/Nxv9arm4Ti en cons
AEP es un proyecto para difundir material anarquista existente. en consulta
Tinc 4 #PremisTuitort per ser cuqui i fer #Rubiadas, dos @GatsMalvats
Tinc 4 PremisTuitort per ser cuqui i fer Rubiadas, dos GatsMalvats
```

Clasificación de la polaridad de un twit

```
In [54]: import nltk
         from nltk import word_tokenize

         #Corpus de entrenamiento

         #Tweets que se consideran de polaridad positiva

         pos_tweets = ['I love this car.', 'This view is amazing.', 'I feel great this morning
                       'I am so excited about the concert.', 'He is my best friend.']

         #Tweets que se consideran de polaridad negativa

         neg_tweets = ['I do not like this car.', 'This view is horrible.', 'I feel tired this
                       'I am not looking forward to the concert.', 'He is my enemy.']

         #Creamos los documentos de entrenamiento: uno para cada tipo opinión

         docs = pos_tweets + neg_tweets

         print("TRAINING TWEETS",docs)

         #Definimos el vectorizador
         from sklearn.feature_extraction.text import TfidfVectorizer
         import numpy as np

         vectorizer = TfidfVectorizer(
             analyzer= 'word',
          )

         #Entrenamiento

         data_labels = []

         for i in range(len(pos_tweets)):
             data_labels.append('POS')                #Data labelling

         for i in range(len(neg_tweets)):
             data_labels.append('NEG')
```

```
        print("TRAINING DATA LABELS", data_labels)

        X = vectorizer.fit_transform(docs)

        X_train = X.toarray()  #Matriz con los vectores de entrenamiento

        from sklearn.naive_bayes import MultinomialNB

        classifier = MultinomialNB()  #Definición del clasificador bayesiano

        bayes_model = classifier.fit(X=X_train, y=data_labels)  #Entrenamiento del clasificad
                                                                #el método al corpus de entre

        #Predicción

        new_tweets = ['Larry is my friend', 'The film is horrible']

        X1 = vectorizer.transform(new_tweets)

        y_pred = bayes_model.predict(X1)

        for i in range(len(new_tweets)):
            print ("LA POLARIDAD DE", new_tweets[i], "ES", y_pred[i])
```

```
TRAINING TWEETS ['I love this car.', 'This view is amazing.', 'I feel great this morning.', 'I
TRAINING DATA LABELS ['POS', 'POS', 'POS', 'POS', 'POS', 'NEG', 'NEG', 'NEG', 'NEG', 'NEG']
LA POLARIDAD DE Larry is my friend ES POS
LA POLARIDAD DE The film is horrible ES NEG
```

1.2.4 1.4 Opinion holder

Obtener el perfil de una cuenta

```
In [55]: import tweepy
         from tweepy import API
         from tweepy import OAuthHandler

         CONSUMER_KEY = XXXXX
         CONSUMER_SECRET = XXXXX
         ACCESS_KEY = XXXXX
         ACCESS_SECRET = XXXXX

         auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
         auth.set_access_token(ACCESS_KEY, ACCESS_SECRET)

         api = tweepy.API(auth)
```

```
description_user = api.get_user('@realDonaldTrump').description

print (description_user)
```

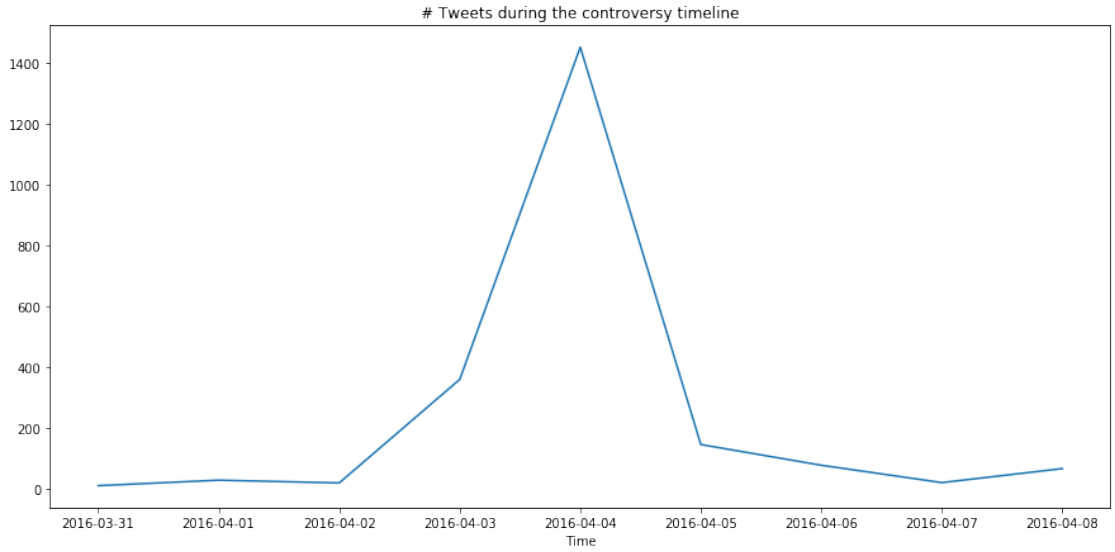
```
---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call last)

<ipython-input-55-a6239ffa0ce8> in <module>
----> 1 import tweepy
      2 from tweepy import API
      3 from tweepy import OAuthHandler
      4
      5 CONSUMER_KEY = XXXXX


ModuleNotFoundError: No module named 'tweepy'
```

1.2.5 1.5 Tiempo

In [56]:
```
#Timeline de los twits publicados sobre la controversia generada por el programa La G

import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime as dt

df =pd.read_csv("lagentnormal-CORPUS.csv", sep='\t', encoding='utf-8')

df1 = df[['Time','User']]

df1['Time'] = pd.to_datetime(df1['Time']).dt.date #Quitar horas y minutos

df1 = df1.groupby('Time').Time.count()
print (df1)

fig, ax = plt.subplots(figsize=(15,7))
df1.plot(title='# Tweets during the controversy timeline')
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  # This is added back by InteractiveShellApp.init_path()
```

```
Time
2016-03-31      10
2016-04-01      28
2016-04-02      19
2016-04-03     359
2016-04-04    1450
2016-04-05     145
2016-04-06      77
2016-04-07      20
2016-04-08      66
Name: Time, dtype: int64
```



In [ ]: