MINNING_TWITTER

June 17, 2019

1 Mineria de datos sobre Twitter y analisis de sentimiento

1.0.1 Importando librerias

Quiza se deban instalar las siguientes librerias en el equipo donde se este trabajando

```
In [ ]: !pip install unidecode
        !pip install cx_Oracle
        !pip install nltk
        !pip install tweepy
```

Importando librerias

```
In [7]: from tweepy import Stream
        from tweepy import OAuthHandler
        from tweepy.streaming import StreamListener
        from html.parser import HTMLParser
        import unidecode
        import itertools
        import cx_Oracle
        import datetime
        import xml.etree.ElementTree as ET
        import json,  re
        import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from wordcloud import WordCloud
        from sklearn.model_selection import train_test_split, RandomizedSearchCV,GridSearchCV
        from sklearn.metrics import classification_report, confusion_matrix
        from scipy.stats import uniform
        from sklearn.pipeline import Pipeline
        import pickle
        %matplotlib inline
```

1.1 Minando data y persistiendola en Oracle

1.1.1 1. Conectanto a la base de datos - Oracle

```
In [ ]: dsn_tns = cx_Oracle.makedsn('???', '1521', service_name='???')
        conn = cx_Oracle.connect(user='????', password='???', dsn=dsn_tns)

        #Hacer commit automaticamente
        conn.autocommit=1

        #Se crea un objeto para ejecutar comandos SQL
        c=conn.cursor()

        #Una variable clob para el formato JSON total
        bvar=c.var(cx_Oracle.CLOB)
```

1.1.2 2. Variables de acceso al Api de Twitter

```
In [ ]: access_token = "??????????????????"
        access_token_secret = "??????????????????"
        consumer_key = "??????????????????"
        consumer_secret = "??????????????????"
```

1.1.3 3. Obteniendo tweets

algunas funciones de limpieza y extraccion

```
In [1]: import re
        hash_regex = re.compile(r"#(\w+)")
        def hash_repl(match):
            _ = match.group(1).upper()
            hstgs.append(_)
            return _

        user_regex = re.compile(r"@(\w+)")
        def user_repl(match):
            _ = match.group(1).upper()
            usr_names.append(_)
            return _

        url_regex = re.compile(r"(http|https|ftp)://[a-zA-Z0-9\./]+")
        def url_repl(match):
            return ''

        def Extraer_metadata(text):
            text = re.sub( hash_regex, hash_repl, text)
            text = re.sub( user_regex, user_repl, text)

        def limpiar_tweet(tweet):
```

```
        html_parser = HTMLParser()
        tweet = html_parser.unescape(tweet)
        tweet = unidecode.unidecode(tweet)
        tweet = tweet.replace("'","")
        tweet = ''.join(''.join(s)[:2] for _, s in itertools.groupby(tweet))#No se uso en e
        tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','', tweet)#No se uso en el minad
        tweet = re.sub('@[^\s]+','', tweet)
        return tweet
```

Minando

```
In [ ]: #Se crea el listener para hacer stream
        class listener(StreamListener):

            def on_data(self, data):
                try:

                    all_data = json.loads(data)
                    tweet = all_data["text"]
                    tweet= limpiar_tweet(tweet)
                    lugar = ""
                    coordenadas = ""
                    geo = ""
                    usuario = all_data["user"]["screen_name"]
                    fecha = all_data["created_at"]
                    if all_data["place"] is not None:
                        lugar = all_data["place"]["name"]
                        coordenadas = all_data["place"]["bounding_box"]["coordinates"]
                        geo = all_data["place"]["bounding_box"]["type"]
                    bvar.setvalue(0,data)
                    try:
                        #se crea el script de insersion
                        sql_str="INSERT INTO AUX_TWITTER_LAB (LLAVE_BUSQUEDA,TWEET,USUARIO,LUG
                        #Ejecutamos script
                        c.execute(sql_str,[bvar])

                    except Exception:
                        sys.exc_clear()
                        print(sql_str)

                    return(True)
                except Exception:
                    sys.exc_clear()
                    print(sql_str)

            def on_error(self, status):
                print(status)
                print(data)
```

```
            print(sql_str)

        #Entrar a Twitter a travez de API
        auth = OAuthHandler(consumer_key, consumer_secret)
        auth.set_access_token(access_token, access_token_secret)
        twitterStream = Stream(auth, listener())

        #Generamos la busqueda de trinos de acuerdo a distintas etiquetas o variables
        twitterStream.filter(track=["Petro,@petrogustavo,Gustavo Petro"],languages=['es'])
```

1.2 Visualizacion y analisis de la data minada

1.2.1 1. Generamos archivos csv para cargar datasets, esta tarea se realiza con base a querys en Oracle SQL

1.2.2 2. Cargamos los archivos CSV a un dataframe de pandas

se uso el delimitador "ñ" pues es un simbolo poco recurrente en trinos y en la estructura del archivo json que retorna el API de twitter

```
In [11]: df_p_rt = pd.read_csv('Data/PETRO_CON_RT.csv',delimiter='ñ',engine='python')
         df_p_nrt = pd.read_csv('Data/PETRO_SIN_RT.csv',delimiter='ñ',engine='python')
         df_u_rt = pd.read_csv('Data/URIBE_CON_RT.csv',delimiter='ñ',engine='python')
         df_u_nrt = pd.read_csv('Data/URIBE_SIN_RT.csv',delimiter='ñ',engine='python')
```

para este ejercicio se trabajaran 4 data sets dos de Gustavo Petro (Un lider de la izquierda politica en Colombia y ex alcalde de Bogota D.C) y de ALvaro Uribe (Expresidente colombiano y un lider de la derecha politica en Colombia).

Se tiene un data set de tweets y un dataset de retweets para cada uno de ellos.

Al parecer la PETRO tambien puede hacer referencia a la criptomoneda en Venezuela y algunos de Uribe refente al desfio 2019 (Un reality) de modo que se eliminan esos tweets pues no aportan a nuestro analisis.

```
In [8]: df_p_nrt = df_p_nrt[df_p_nrt['TWEET'].str.contains('blockchain', na=False)== False]
        df_p_nrt = df_p_nrt[df_p_nrt['TWEET'].str.contains('VenezuelaEsCultura2019', na=False)=
        df_p_rt = df_p_rt[df_p_rt['TWEET'].str.contains('blockchain', na=False)== False]
        df_p_rt = df_p_rt[df_p_rt['TWEET'].str.contains('VenezuelaEsCultura2019', na=False)== I

        df_u_nrt = df_u_nrt[df_u_nrt['TWEET'].str.contains('desafiosuperregiones', na=False)==
        df_u_rt = df_u_rt[df_u_rt['TWEET'].str.contains('desafiosuperregiones', na=False)== Fal
```

```
In [9]: print("La cantidad de tweets recolectados con referencia a Petro es de : {}".format(df_
        print("La cantidad de retweets recolectados con referencia a Petro es de : {}".format(d
        print("La cantidad de tweets recolectados con referencia a Uribe es de : {}".format(df_
        print("La cantidad de retweets recolectados con referencia a Uribe es de : {}".format(d
```

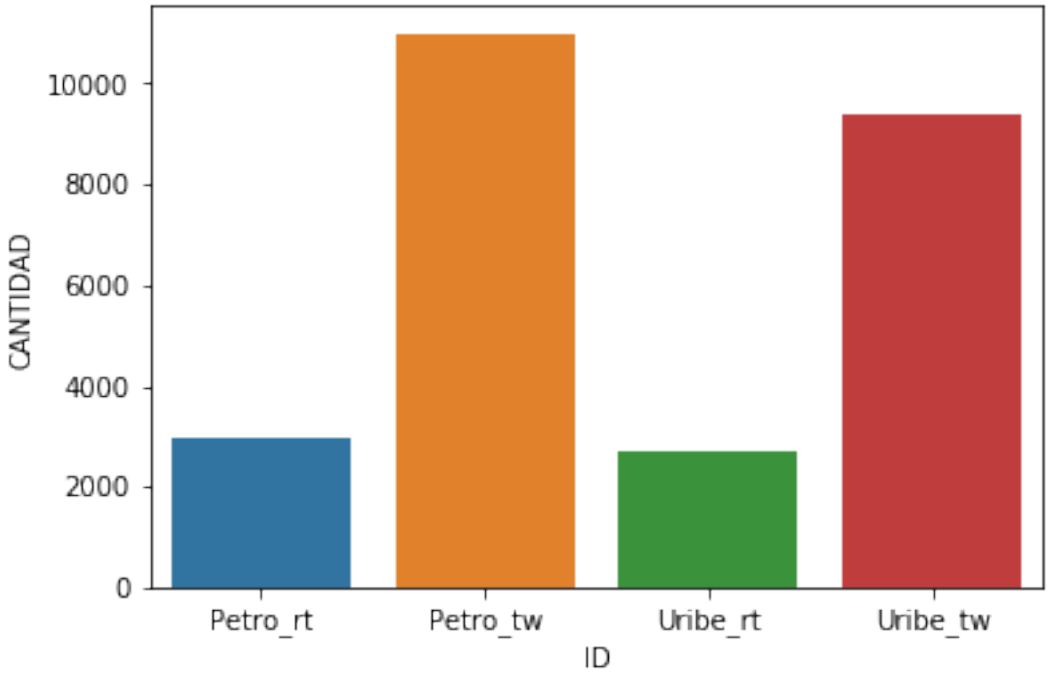
```
La cantidad de tweets recolectados con referencia a Petro es de : 10987
La cantidad de retweets recolectados con referencia a Petro es de : 2957
La cantidad de tweets recolectados con referencia a Uribe es de : 9382
La cantidad de retweets recolectados con referencia a Uribe es de : 2726
```

```sql
create table aux_twitter_lab(
llave_busqueda varchar2(50),
tweet varchar2(500),
usuario varchar2(100),
fecha TIMESTAMP (6) DEFAULT systimestamp,
lugar varchar2(100),
coordenadas varchar2(200),
geo varchar2(100),
json clob)
tablespace ts_table_m;

--*****************************************************************************
--*** Consulta con el fin de identificar la cantidad de tweets por personaje ***
--*****************************************************************************
select llave_busqueda,count(0) conteo from aux_twitter_lab
group by llave_busqueda;

--*************************************************************
--*** Consulta para treaer todo los trinos que no sean reweet ***
--*************************************************************
select
LLAVE_BUSQUEDA,
replace(replace(tweet,chr(10),''),chr(13),'') tweet,
USUARIO,
FECHA,
LUGAR,
COORDENADAS,
GEO
from aux_twitter_lab
where llave_busqueda = 'URIBE'
and tweet NOT like 'RT %';

--*************************************************************
--*** Consulta para treaer todo los trinos que  sean reweet ***
--*************************************************************
select
LLAVE_BUSQUEDA,
replace(replace(tweet,chr(10),''),chr(13),'') tweet,
SUBSTR(tweet,3,INSTR(tweet, ':')-3) USUARIO_RT,
COUNT(0) CONTEO
from aux_twitter_lab
where llave_busqueda = 'PETRO'
and tweet like 'RT %'
AND SUBSTR(tweet,3,INSTR(tweet, ':')-3) IS NOT NULL
GROUP BY LLAVE_BUSQUEDA,
replace(replace(tweet,chr(10),''),chr(13),'') ,
SUBSTR(tweet,3,INSTR(tweet, ':')-3)
ORDER BY 4 DESC;
```

title

En una cantidad similar de tiempo parece que ambos lideres politicas tienen una cantidad muy parecida de tweets y retweets, teniendo Petro una leve ventaja en menciones.

```
In [10]: d = {'ID': ["Petro_rt", "Petro_tw","Uribe_rt","Uribe_tw"], 'CANTIDAD': [df_p_rt.count
         df_cantidades = pd.DataFrame(d)
         sns.barplot(x='ID',y='CANTIDAD',data=df_cantidades)
         plt.show()
```



1.2.3 3. Analisis y visualizaciones

Extraemos metadata de los tweets (Tendencias y usuarios)

1.2.4 Gustavo Petro tweets

```
In [195]: hstgs = [] # Extraer hashtags
          usr_names = [] # Extraer nombres de usuario

          df_p_nrt.TWEET.apply(Extraer_metadata)
          petro_hash = WordCloud(background_color="white",stopwords=['petro']).generate(" ".jo
          petro_user = WordCloud(background_color="white",stopwords=['petrogustavo']).generate

          fig, axs = plt.subplots(2, 1,figsize=(20,15))
          plt.subplot(2,1,1).set_title("WordCloud para tendencias usadas al mencionar a Petro"
          plt.imshow(petro_hash, interpolation='bilinear')
          plt.axis("off")
```

```
        plt.subplot(2,1,2).set_title("WordCloud para usuarios que hablan de Petro")
        plt.imshow(petro_user, interpolation='bilinear')
        plt.axis("off")
```

```
Out[195]: (-0.5, 399.5, 199.5, -0.5)
```

1.2.5 Gustavo Petro retweets

```
In [15]: hstgs = [] # Extraer hashtags
         usr_names = [] # Extraer nombres de usuario
```

```
df_p_rt.TWEET.apply(Extraer_metadata)
petro_hash = WordCloud(background_color="white",stopwords=['petro']).generate(" ".joi
petro_user = WordCloud(background_color="white",stopwords=['petrogustavo']).generate(

fig, axs = plt.subplots(2, 1,figsize=(20,15))
plt.subplot(2,1,1).set_title("WordCloud para tendencias usadas al mencionar a Petro y
plt.imshow(petro_hash, interpolation='bilinear')
plt.axis("off")
plt.subplot(2,1,2).set_title("WordCloud para usuarios que hablan de Petro y tienen re
plt.imshow(petro_user, interpolation='bilinear')
plt.axis("off")
```

```
Out[15]: (-0.5, 399.5, 199.5, -0.5)
```

1.2.6 Alvaro Uribe tweets

```
In [16]: hstgs = [] # Extraer hashtags
         usr_names = [] # Extraer nombres de usuario

         df_u_nrt.TWEET.apply(Extraer_metadata)
         petro_hash = WordCloud(background_color="white",stopwords=['uribe']).generate(" ".joi
         petro_user = WordCloud(background_color="white",stopwords=['alvarouribevel']).generate

         fig, axs = plt.subplots(2, 1,figsize=(20,15))
```

```
plt.subplot(2,1,1).set_title("WordCloud para tendencias usadas al mencionar a Uribe")
plt.imshow(petro_hash, interpolation='bilinear')
plt.axis("off")
plt.subplot(2,1,2).set_title("WordCloud para usuarios que hablan de Uribe")
plt.imshow(petro_user, interpolation='bilinear')
plt.axis("off")
```

Out[16]: (-0.5, 399.5, 199.5, -0.5)

1.2.7 Alvaro Uribe retweets

In [17]:
```
hstgs = [] # Extraer hashtags
usr_names = [] # Extraer nombres de usuario

df_u_rt.TWEET.apply(Extraer_metadata)
petro_hash = WordCloud(background_color="white",stopwords=['uribe']).generate(" ".joi
petro_user = WordCloud(background_color="white",stopwords=['alvarouribevel']).generat

fig, axs = plt.subplots(2, 1,figsize=(20,15))
plt.subplot(2,1,1).set_title("WordCloud para tendencias usadas al mencionar a Uribe y
plt.imshow(petro_hash, interpolation='bilinear')
plt.axis("off")
plt.subplot(2,1,2).set_title("WordCloud para usuarios que hablan de Uribe y tienen re
plt.imshow(petro_user, interpolation='bilinear')
plt.axis("off")
```

Out[17]:
```
(-0.5, 399.5, 199.5, -0.5)
```

1.3 Hallazgos

1.3.1 Petro:

1. Tweets:

1.1 Las dos mas importantes tendencias referentes a Petro son: FIRMOPARAQUEURIBESERETIRE y ENELEXTERIORDEFENDEMOSLAPAZ los cuales son tendencias en donde en su mayoria apoyan a Petro.

1.2 Sin envargo las tendencias mas relevantes que le sigue son en contra de Petro:(LLORATONMAMERTA,SANTRICHNARCONGRESISTA y LABOLSAPETRO)

1.3 Los usuarios que mas se mencionan con respecto a Petro se comportan asi:
-KSTROJK (Detractor)
-ENRIQUEPENALOSA (Detractor)
-MARIAFDACABAL (Detractor)
-GUSTAVOBOLIVAR (A favor)
-IVANCEPEDACAST (A favor)
-AIDAAVELLAE (A favor)
-DEFENDAMOSPAZ (A favor)

Por lo tanto en cuanto a usuarios la distribucion de detractores y personas a favor es muy similar.

2. Retweets:

2.1 Las tendencias en cuanto a retweets se comportan de una manera muy similar a la de los tweets.

1.3 Los usuarios a los que mas retweetean cuando se habla de Petro se comportan asi:
-MARIAFDACABAL (Detractor)
-GUSTAVOBOLIVAR (A favor)
-AIDAAVELLAE (A favor)
-IVANCEPEDACAST (A favor)
-KSTROJK (Detractor)
Luego le siguen:
-DEFENDAMOSPAZ (A favor)
-MLUCIARAMIREZ (Detractor)
-CRISTOBUSTOS (Detractor)
-ALVAROURIBEVEL (Detractor)

Se puede observar que cuentas como la de ENRIQUEPENALOSA, IVANDUQUE, JEROBLEDO, CLAUDIALOPEZ, ALVAROURIBEVEL tambien son retweeteadas pero no en el mismo nivel que cuentas anteriores.

1.3.2 Uribe

1. Tweets:

1.1 Las dos mas importantes tendencias referentes a Uribe son: NODESPRESTIGIENLASCORTES,NARCO,URIBENARCO82, SANTRICH, ALAIRE cuales son tendencias en donde en su mayoria atacan al senador Uribe exceptuando a SANTRICH y ALAIRE que son tendencias neutras.

1.2 tambien existen dos tendencias relevantes SISALESANTRICHSALEURIBE y NYTIMESFAKENEWS en donde una tendencia nuevamente es en contra del senador y otra a favor respectivamente

1.3 Los usuarios que mas se mencionan con respecto a Uribe se comportan asi:
-ANGELAMROBLEDO (Detractor)
-IVANDUQUE (A favor)
-INTIASPRILLA (Detractor)
-GUSTAVOBOLIVAR (Detractor)
-MIRANDABOGOTA (Detractor)
-CEDEMOCRATICO (A favor)

La mayoria de menciones en los tweets relacionan a personas en contro de Uribe.

2. Retweets:

2.1 Las tendencias en cuanto a retweets se comportan de una manera muy similar a la de los tweets (En su mayoria negativas).

1.3 Los usuarios a los que mas retweetean cuando se habla de Uribe se comportan asi:
-IVANDUQUE (A favor)
-CEDEMOCRATICO (A favor)
-INTIASPRILLA (Detractor)
-MARGARITAREPO (A favor)
-MIRANDABOGOTA (Detractor)
-ANGELAMROBLEDO (Detractor)
-GUSTAVOBOLIVAR (Detractor)
Luego le siguen:
-DANIELSAMPERO (Detractor)
-SENORCAISEDO (Detractor)
-JUANITAGOE (Detractor)
-YUMBILA (Detractor)
-CARLOSCARRILLOA (Detractor)
-PETROGUSTAVO (Detractor)
-GABRIELSANTOSCD (A favor)

Se puede observar que cuentas la mayoria de personas a las que hacen retweet son detractores o personas en contra de Uribe.

1.4 Cargando y modificando nuestro dataset de entrenamiento

1.4.1 1. Cargamos los archivos .xml con los tweets y su clasificación

Pasando a un data set de pandas el archivo general-train-tagged-3l.xml y politics2013-tweets-test-tagged.xml

```
In [18]: row=0
         df_train = pd.DataFrame({'tweet_id':[],'tweetText':[],'polarity_value':[],'polarity_ty
         archivos = ["Data/general-train-tagged-3l.xml", "Data/politics2013-tweets-test-tagged
         for arc_xml in archivos:
          tree = ET.parse(arc_xml)
          root = tree.getroot()
          for tweet in root:
              tweet_id = 'ID:'+tweet.find('tweetid').text
              tweetText = tweet.find('content').text
              lang = tweet.find('lang').text
              polarity_value = tweet.find('sentiments').find('polarity').find('value').text
              polarity_type = tweet.find('sentiments').find('polarity').find('type').text
              topic = tweet.find('topics').find('topic').text

              if lang == 'es':
                  df_train.loc[row] = [tweet_id,tweetText,polarity_value,polarity_type,topic]
                  row+=1

In [19]: df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9718 entries, 0 to 9717
Data columns (total 5 columns):
tweet_id          9718 non-null object
tweetText         9718 non-null object
polarity_value    9718 non-null object
polarity_type     9718 non-null object
topic             9718 non-null object
dtypes: object(5)
memory usage: 455.5+ KB
```

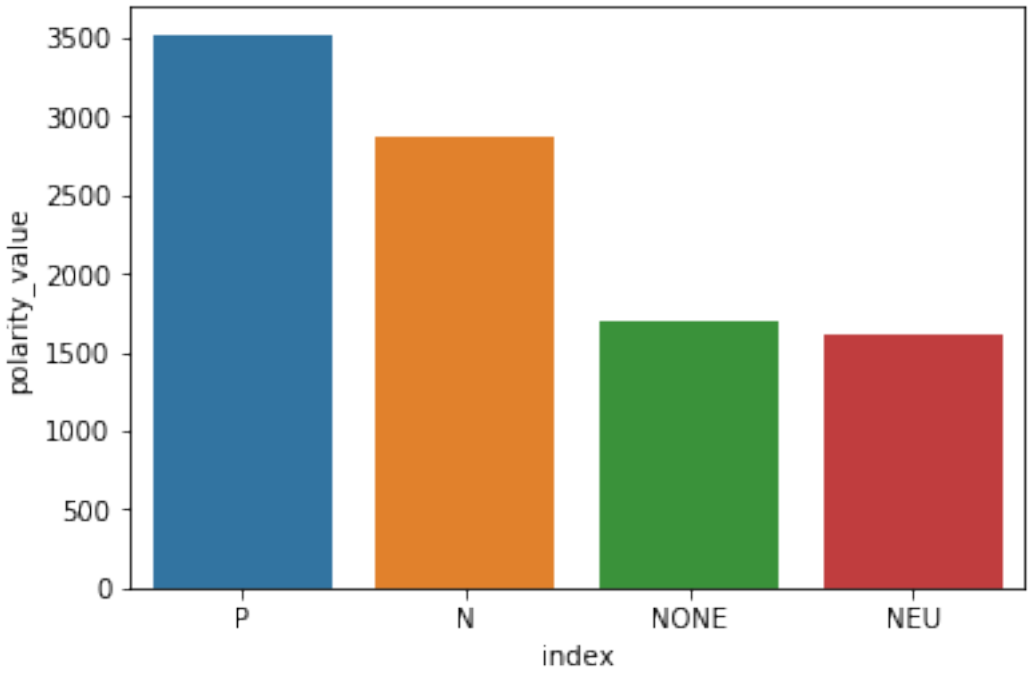
1.4.2 2. Analizamos mediante visualizaciones nuestros datos de entrenamiento

Realizamos analisis exploratorio de nuestro data set de entrenamiento en cuanto al comportamiento de las etiquetas:

-P : Positivo
-N: Negativo
-None: No aplica
-NEU: Neutral

```
In [20]: df_etiqueta=pd.DataFrame(df_train['polarity_value'].value_counts())
         df_etiqueta = df_etiqueta.reset_index()
```

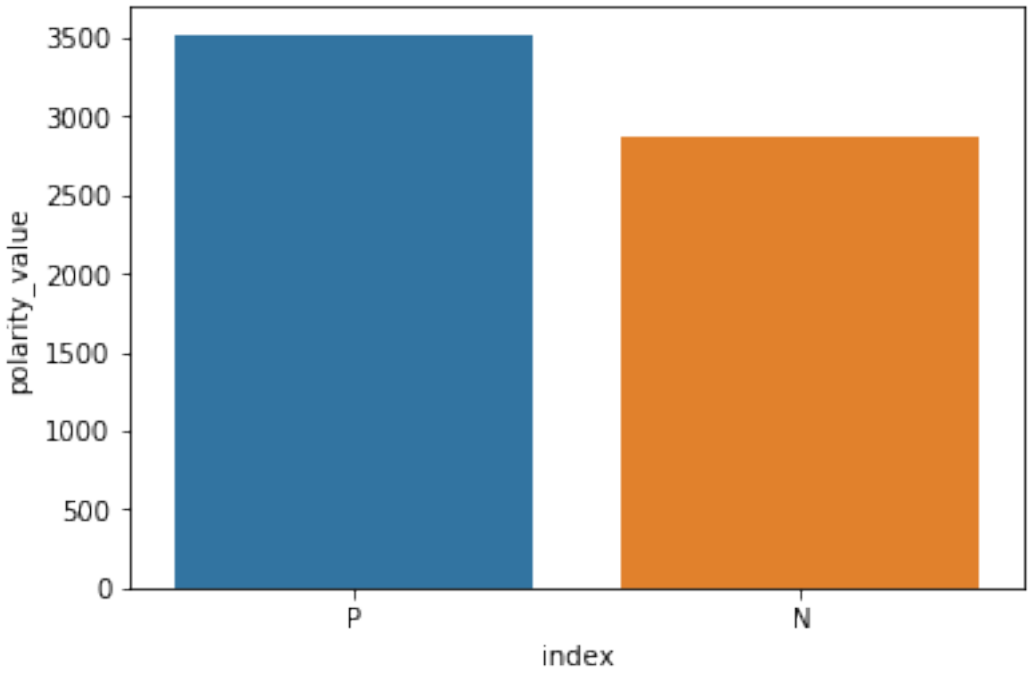
```
In [21]: sns.barplot(x='index',y='polarity_value',data=df_etiqueta)
         plt.show()
```



Eliminemos los ejemplos con etiqueta NONE y NEU para que no entorpezca los analisis

```
In [22]: df_train = df_train[df_train['polarity_value'].isin(['P','N'])]
         df_etiqueta=pd.DataFrame(df_train['polarity_value'].value_counts())
         df_etiqueta = df_etiqueta.reset_index()
```

```
In [23]: sns.barplot(x='index',y='polarity_value',data=df_etiqueta)
         plt.show()
```



Las graficas de **wordcloud** se comportan muy bien en para analizar cadenas de texto.

En este caso eliminamos palabras que no aportan mucho al mensaje o texto y palabras muy especificas al gobierno de España que no seran utiles para el caso de Colombia

```
In [3]: import nltk
        from nltk.corpus import stopwords
        nltk.download('stopwords')
        nltk.download('punkt')
        spanish_stopwords = stopwords.words('spanish')
        spanish_stopwords.append('rt')
        spanish_stopwords.append('http')
        spanish_stopwords.append('co')
        spanish_stopwords.append('upyd')
        spanish_stopwords.append('pp')
        spanish_stopwords.append('20n')
```

```
        spanish_stopwords.append('rajoy')
        spanish_stopwords.append('psoe')
        spanish_stopwords.append('espana')
        spanish_stopwords.append('user')
        spanish_stopwords.append('url')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/alarconc/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/alarconc/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Realizamos la limpieza inicial con la que descargamos la data de Twitter al conjunto de entrenamiento

```
In [25]: df_train['Tweet'] = df_train['tweetText'].apply(limpiar_tweet)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24: DeprecationWarni
```

Divimos la data en cada uno de nuestras 3 etiquetas P,N y NEU para crear wordcloud para cada una

```
In [26]: df_train_pos = df_train[df_train['polarity_value']=='P']['Tweet']
         df_train_neg = df_train[df_train['polarity_value']=='N']['Tweet']


         wordcloudpos = WordCloud(stopwords=spanish_stopwords, background_color="white").genera

         wordcloudneg = WordCloud(stopwords=spanish_stopwords, background_color="white").genera


         fig, axs = plt.subplots(1, 2,figsize=(20,10))

         # Display the generated image:
         #plt.figure(figsize=(20,10))
         plt.subplot(2,2,1).set_title("WordCloud para sentimientos positivos")
         plt.imshow(wordcloudpos, interpolation='bilinear')
         plt.axis("off")
         plt.subplot(2,2,2).set_title("WordCloud para sentimientos negativos")
         plt.imshow(wordcloudneg, interpolation='bilinear')
         plt.axis("off")
```

```
Out[26]: (-0.5, 399.5, 199.5, -0.5)
```

1.4.3 3. Nuestra data para entrenamiento es de tipo TEXTO y la gran mayoria de modelos de machine learning funcionan con data numerica, de modo que hace falta un proceso total de limpieza y transformacion de los datos al que llamaremos "Vectorizar"

In [4]:
```
#Importando librerias
import nltk
from nltk import word_tokenize
from nltk.data import load
from nltk.stem import SnowballStemmer
from string import punctuation
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer, TfidfTrans
nltk.download('stopwords')
nltk.download('punkt')

# Modelo de stemmer que se usa para normalizar las palabras, tipo:
#Playing = play
#Player = play
#Plyed  = play

stemmer = SnowballStemmer('spanish')

#Eliminando signos de puntuacion
non_words = list(punctuation)
non_words.extend(['£', 'ą'])
non_words.extend(map(str,range(10)))

#Aplicando stemmer a cada token
def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

#Tokenizar se le llama al proceso de separar cada frase en palabras individuales.
def tokenize(text):
    text = ''.join([c for c in text if c not in non_words])
```

```
            tokens =  word_tokenize(text)

            #stem
            try:
                stems = stem_tokens(tokens, stemmer)
            except Exception as e:
                print(e)
                print(text)
                stems = ['']
            return stems

        #Con un objeto de tipo TfidfVectorizer se le asigna la tokenizacion que se desarrollo
        #Todo esto se guarda en un objeto de tipo TfidfVectorizer llamado vectorizer, el cual
        count_vectorizer = CountVectorizer(
                        analyzer = 'word',
                        tokenizer = tokenize,
                        stop_words = spanish_stopwords,
                        lowercase = True
                        )
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/alarconc/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/alarconc/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

1.4.4 3. Buscando distintos modelos de clasificacion para entrenar nuestra data

Separamos nuestro conjunto de entrenamiento en un pocentaje de 70-30 para entrenamiento y test respectivamente

```
In [29]: X = df_train['Tweet']
         y = df_train['polarity_value']
         Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,stratify=y,test_size=0.3,random_st
```

Naive_bayes - MultinomialNB

```
In [34]: from sklearn.naive_bayes import MultinomialNB

         text_clf = Pipeline([('vect', count_vectorizer),
                              ('tfidf', TfidfTransformer()),
                              ('clf', MultinomialNB())])

         tuned_parameters = {
             'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
             'tfidf__use_idf': (True, False),
             'tfidf__norm': ('l1', 'l2'),
```

```
            'clf__alpha': np.linspace(0.5, 1.5, 6)}

        clf = GridSearchCV(text_clf, tuned_parameters, cv=10, n_jobs=5)
        clf.fit(Xtrain, ytrain)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:300: Us
  'stop_words.' % sorted(inconsistent))
```

```
Out[34]: GridSearchCV(cv=10, error_score='raise-deprecating',
               estimator=Pipeline(memory=None,
              steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='stri
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None,
                 stop_words=['de', 'la'...inear_tf=False, use_idf=True)), ('clf', MultinomialNB
               fit_params=None, iid='warn', n_jobs=5,
               param_grid={'vect__ngram_range': [(1, 1), (1, 2), (2, 2)], 'tfidf__use_idf': (
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring=None, verbose=0)
```

Asignamos el modelo con la mejor combinacion de parametros

```
In [35]: best_baye_clf = clf.best_estimator_
         y_pred = best_baye_clf.predict(Xtest)
```

```
In [36]: print("El mejor scoring para el modelo es: {} ".format(clf.best_score_))
         print("El score obtenido para el set de test es: {}".format(best_baye_clf.score(Xtest
         print("El mejor modelo tiene estos parametros: {}".format(clf.best_params_))
         print("Classification report:")
         print(classification_report(ytest,y_pred))
         print("Matriz de confusion:")
         print(confusion_matrix(ytest,y_pred))
```

```
El mejor scoring para el modelo es: 0.7610441767068273
El score obtenido para el set de test es: 0.7527329515877147
El mejor modelo tiene estos parametros: {'clf__alpha': 0.5, 'tfidf__norm': 'l2', 'tfidf__use_id
Classification report:
              precision    recall  f1-score   support

           N       0.76      0.65      0.70       864
           P       0.75      0.83      0.79      1057

   micro avg       0.75      0.75      0.75      1921
   macro avg       0.75      0.74      0.75      1921
weighted avg       0.75      0.75      0.75      1921

Matriz de confusion:
[[565 299]
```

```
 [176 881]]
```

```
In [37]: # save the model to disk
         filename = 'Modelos/best_baye_clf.sav'
         pickle.dump(best_baye_clf, open(filename, 'wb'))
```

Random Forest

```
In [38]: from sklearn.ensemble import RandomForestClassifier

         n_estimators = [int(x) for x in np.linspace(start = 200, stop = 3000, num = 10)]
         max_features = ['auto', 'sqrt']
         max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
         max_depth.append(None)
         min_samples_split = [2, 5, 10]
         min_samples_leaf = [1, 2, 4]
         bootstrap = [True, False]


         text_clf_2 = Pipeline([('vect', count_vectorizer),
                                ('tfidf', TfidfTransformer()),
                                ('clf', RandomForestClassifier())])

         tuned_parameters_2 = {
             'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
             'tfidf__use_idf': (True, False),
             'tfidf__norm': ('l1', 'l2'),
             'clf__n_estimators' : n_estimators,
             'clf__max_features' : max_features,
             'clf__max_depth': max_depth,
             'clf__min_samples_split': min_samples_split,
             'clf__min_samples_leaf': min_samples_leaf,
             'clf__bootstrap': bootstrap}

         clf_2 = RandomizedSearchCV(text_clf_2, tuned_parameters_2, cv=10, n_jobs=5, n_iter = 4
         clf_2.fit(Xtrain, ytrain)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/pr
  "timeout or by a memory leak.", UserWarning
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/pr
  "timeout or by a memory leak.", UserWarning
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:300: Us
  'stop_words.' % sorted(inconsistent))
```

```
Out[38]: RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                    estimator=Pipeline(memory=None,
```

```
                steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='stri
                   dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                   lowercase=True, max_df=1.0, max_features=None, min_df=1,
                   ngram_range=(1, 1), preprocessor=None,
                   stop_words=['de', 'la'...obs=None,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False))]),
                  fit_params=None, iid='warn', n_iter=40, n_jobs=5,
                  param_distributions={'vect__ngram_range': [(1, 1), (1, 2), (2, 2)], 'tfidf_
                  pre_dispatch='2*n_jobs', random_state=None, refit=True,
                  return_train_score='warn', scoring=None, verbose=0)
```

```
In [39]: best_rnd_forest_clf =clf_2.best_estimator_
         y_pred = best_rnd_forest_clf.predict(Xtest)
```

```
In [40]: print("El mejor scoring para el modelo es: {} ".format(clf_2.best_score_))
         print("El score obtenido para el set de test es: {}".format(best_rnd_forest_clf.score
         print("El mejor modelo tiene estos parametros: {}".format(clf_2.best_params_))
         print("Classification report:")
         print(classification_report(ytest,y_pred))
         print("Matriz de confusion:")
         print(confusion_matrix(ytest,y_pred))
```

```
El mejor scoring para el modelo es: 0.7327086122266845
El score obtenido para el set de test es: 0.7147319104633003
El mejor modelo tiene estos parametros: {'vect__ngram_range': (1, 1), 'tfidf__use_idf': False,
Classification report:
              precision    recall  f1-score   support

           N       0.72      0.59      0.65       864
           P       0.71      0.81      0.76      1057

   micro avg       0.71      0.71      0.71      1921
   macro avg       0.72      0.70      0.71      1921
weighted avg       0.72      0.71      0.71      1921

Matriz de confusion:
[[514 350]
 [198 859]]
```

```
In [41]: # save the model to disk
         filename = 'Modelos/best_rnd_forest_clf.sav'
         pickle.dump(best_rnd_forest_clf, open(filename, 'wb'))
```

Suport Vector Machine - SVC

```
In [42]: from sklearn.svm import SVC
```

```
Cs = [0.001, 0.01, 0.1, 1, 10]
gammas = [0.001, 0.01, 0.1, 1]
kernels = ['rbf','sigmoid','linear']


text_clf_3 = Pipeline([('vect', count_vectorizer),
                       ('tfidf', TfidfTransformer()),
                       ('clf', SVC())])



tuned_parameters_3 = {
    'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
    'tfidf__use_idf': (True, False),
    'tfidf__norm': ('l1', 'l2'),
    'clf__C': Cs,
    'clf__gamma':gammas,
    'clf__kernel':kernels}

clf_3 = GridSearchCV(text_clf_3, tuned_parameters_3, cv=10, n_jobs=5)
clf_3.fit(Xtrain, ytrain)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:300: Us
  'stop_words.' % sorted(inconsistent))
```

```
Out[42]: GridSearchCV(cv=10, error_score='raise-deprecating',
               estimator=Pipeline(memory=None,
              steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='str
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None,
                 stop_words=['de', 'la'...f', max_iter=-1, probability=False, random_state=None
            shrinking=True, tol=0.001, verbose=False))]),
               fit_params=None, iid='warn', n_jobs=5,
               param_grid={'vect__ngram_range': [(1, 1), (1, 2), (2, 2)], 'tfidf__use_idf': (
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring=None, verbose=0)
```

```
In [43]: best_svc_clf = clf_3.best_estimator_
         y_pred = best_svc_clf.predict(Xtest)
```

```
In [44]: print("El mejor scoring para el modelo es: {} ".format(clf_3.best_score_))
         print("El score obtenido para el set de test es: {}".format(best_svc_clf.score(Xtest,y
         print("El mejor modelo tiene estos parametros: {}".format(clf_3.best_params_))
         print("Classification report:")
         print(classification_report(ytest,y_pred))
         print("Matriz de confusion:")
         print(confusion_matrix(ytest,y_pred))
```

```
El mejor scoring para el modelo es: 0.7657295850066934
El score obtenido para el set de test es: 0.7537740760020822
El mejor modelo tiene estos parametros: {'clf__C': 1, 'clf__gamma': 1, 'clf__kernel': 'rbf', '
Classification report:
              precision    recall  f1-score   support

           N       0.74      0.71      0.72       864
           P       0.77      0.79      0.78      1057

   micro avg       0.75      0.75      0.75      1921
   macro avg       0.75      0.75      0.75      1921
weighted avg       0.75      0.75      0.75      1921

Matriz de confusion:
[[611 253]
 [220 837]]
```

```
In [45]: # save the model to disk
         filename = 'Modelos/best_svc_clf.sav'
         pickle.dump(best_svc_clf, open(filename, 'wb'))
```

LogisticRegression

```
In [47]: from sklearn.linear_model import LogisticRegression


         text_clf_4 = Pipeline([('vect', count_vectorizer),
                                ('tfidf', TfidfTransformer()),
                                ('clf', LogisticRegression())])



         tuned_parameters_4 = {
             'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
             'tfidf__use_idf': (True, False),
             'tfidf__norm': ('l1', 'l2'),
             'clf__penalty': ['l1', 'l2'],
             'clf__C': [1, 5, 10],
             'clf__max_iter': [20, 50, 100]}

         clf_4 = GridSearchCV(text_clf_4, tuned_parameters_4, cv=10, n_jobs=5)
         clf_4.fit(Xtrain, ytrain)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:300: Us
  'stop_words.' % sorted(inconsistent))
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: Futu
  FutureWarning)
```

```
Out[47]: GridSearchCV(cv=10, error_score='raise-deprecating',
                estimator=Pipeline(memory=None,
               steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='str
                  dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                  lowercase=True, max_df=1.0, max_features=None, min_df=1,
                  ngram_range=(1, 1), preprocessor=None,
                  stop_words=['de', 'la'...penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False))]),
               fit_params=None, iid='warn', n_jobs=5,
               param_grid={'vect__ngram_range': [(1, 1), (1, 2), (2, 2)], 'tfidf__use_idf': (
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring=None, verbose=0)
```

```
In [48]: best_lr_clf = clf_4.best_estimator_
         y_pred = best_lr_clf.predict(Xtest)
```

```
In [49]: print("El mejor scoring para el modelo es: {} ".format(clf_4.best_score_))
         print("El score obtenido para el set de test es: {}".format(best_lr_clf.score(Xtest,y
         print("El mejor modelo tiene estos parametros: {}".format(clf_4.best_params_))
         print("Classification report:")
         print(classification_report(ytest,y_pred))
         print("Matriz de confusion:")
         print(confusion_matrix(ytest,y_pred))
```

```
El mejor scoring para el modelo es: 0.7648371262829095
El score obtenido para el set de test es: 0.7480478917230609
El mejor modelo tiene estos parametros: {'clf__C': 10, 'clf__max_iter': 20, 'clf__penalty': 'l2
Classification report:
              precision    recall  f1-score   support

           N       0.73      0.71      0.72       864
           P       0.77      0.78      0.77      1057

   micro avg       0.75      0.75      0.75      1921
   macro avg       0.75      0.74      0.74      1921
weighted avg       0.75      0.75      0.75      1921

Matriz de confusion:
[[612 252]
 [232 825]]
```

```
In [50]: # save the model to disk
         filename = 'Modelos/best_lr_clf.sav'
         pickle.dump(best_lr_clf, open(filename, 'wb'))
```

1.4.5 4. Seleccion del modelo basados en el score de la data test

```
In [57]: d = {'Modelo': ["MultinomialNB", "RandomForest","SVC","LogisticRegression"],
              'Score': [clf.best_score_, clf_2.best_score_,clf_3.best_score_,clf_4.best_score_
```

```
        'Score_test':[clf.best_estimator_.score(Xtest,ytest),clf_2.best_estimator_.score
Modelos = pd.DataFrame(d)
Modelos.sort_values('Score_test', ascending=False)
```

Out[57]:

	Modelo	Score	Score_test
2	SVC	0.765730	0.753774
0	MultinomialNB	0.761044	0.752733
3	LogisticRegression	0.764837	0.748048
1	RandomForest	0.732709	0.714732

Al parecer el mejor modelo es Svc y seguido por muy poco MultinomialNB de modo que realizaremos el ejercicio con estos dos modelos.

Tambien hace falta resaltar que un score de 75% no es muy bueno

1.5 Usando el modelo entrenado

```
In [10]: #Dado el caso que se requiera cargar el modelo ya entrenado
         best_svc_clf = pickle.load(open('Modelos/best_lr_clf.sav', 'rb'))
```

1.6 1. Preparamos dataset

```
In [12]: df_total = pd.concat([df_p_nrt,df_u_nrt])
```

```
In [13]: X_def = df_total['TWEET'].apply(limpiar_tweet)
         y_def = best_svc_clf.predict(X_def)
```

```
/home/alarconc/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24: DeprecationWarni
/home/alarconc/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:300: Us
  'stop_words.' % sorted(inconsistent))
```

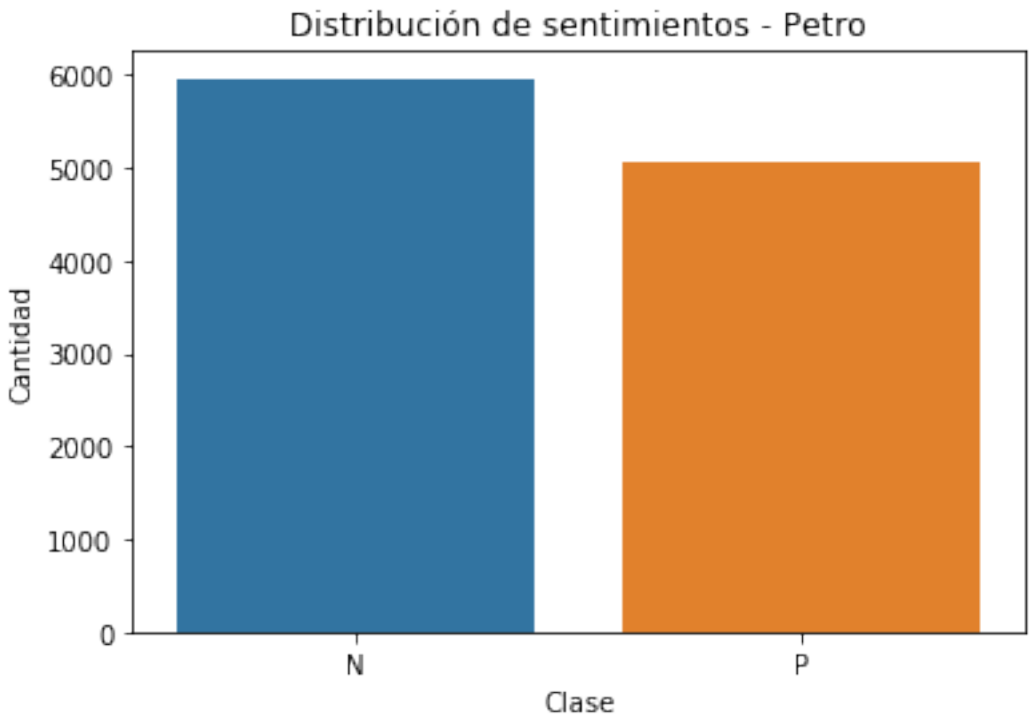
```
In [17]: df_total['class'] = y_def
```

```
In [18]: DF_PETRO = df_total[df_total['LLAVE_BUSQUEDA']=='PETRO']
         DF_URIBE = df_total[df_total['LLAVE_BUSQUEDA']=='URIBE']
```

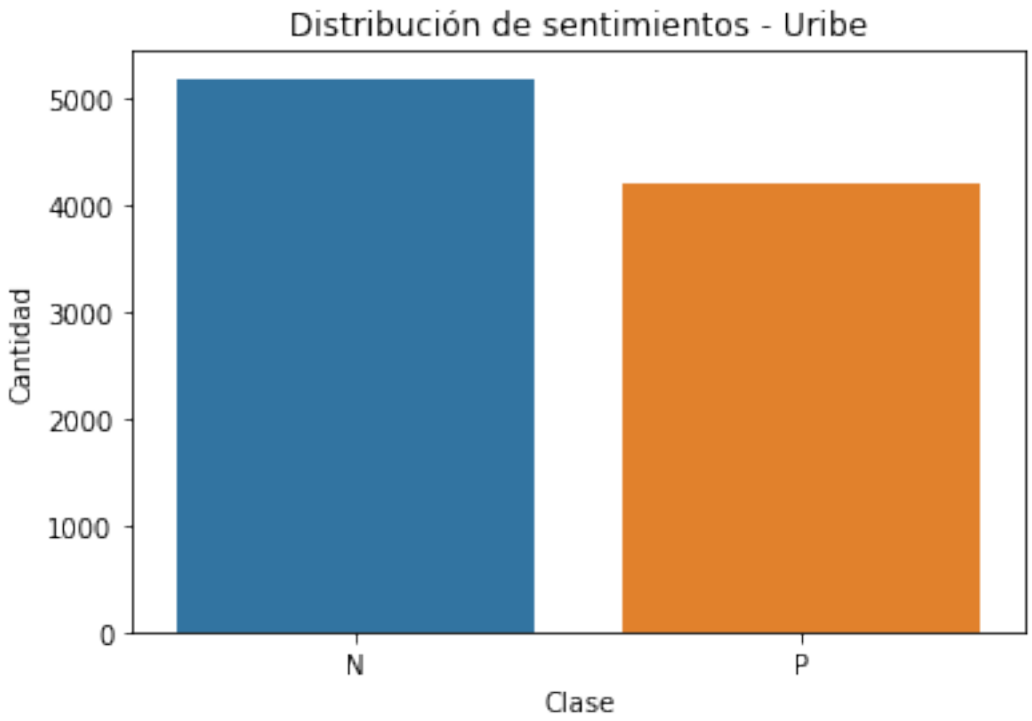
1.7 2. Visualizacion de los tweets clasificados

1.7.1 Diagramas de barra para el comportamiento de la clasificacion en cada tweet

```
In [19]: df_class_petro=pd.DataFrame(DF_PETRO['class'].value_counts())
         df_class_petro = df_class_petro.reset_index()
         sns.barplot(x='index',y='class',data=df_class_petro)
         plt.xlabel('Clase')
         plt.ylabel('Cantidad')
         plt.title('Distribución de sentimientos - Petro')
         plt.show()
```

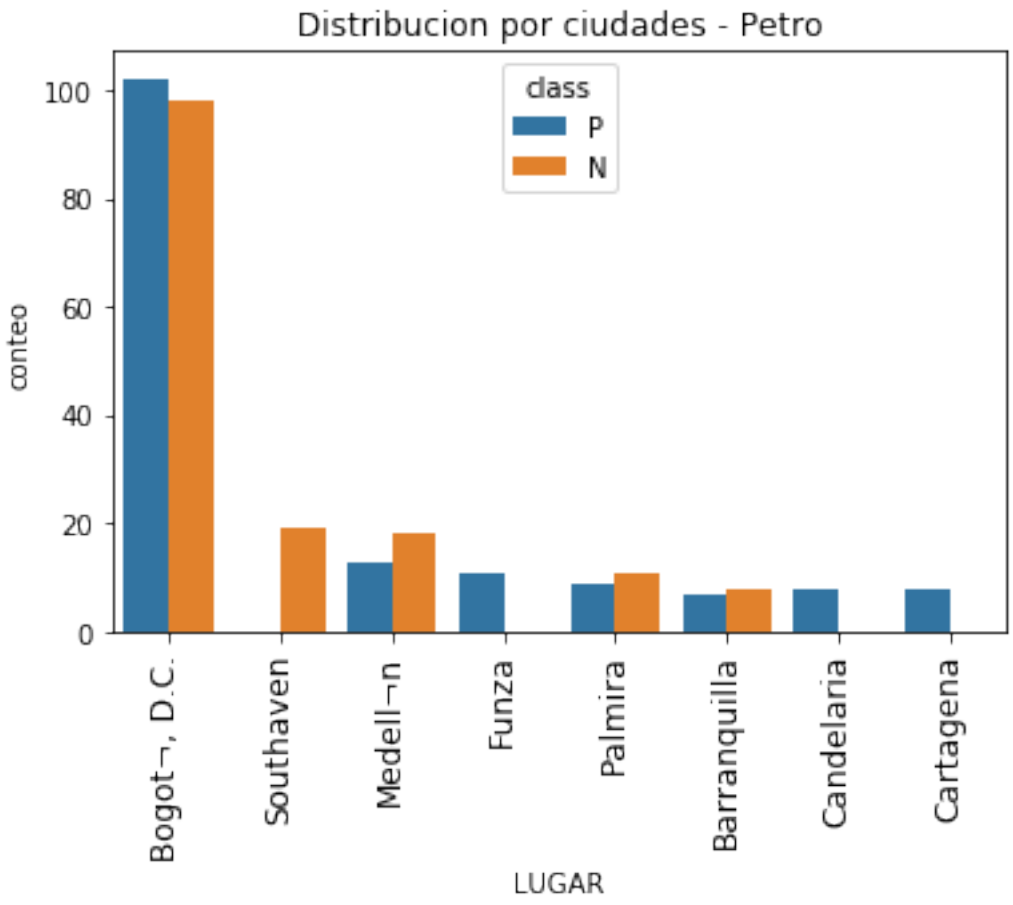



```
In [20]: df_class_uribe=pd.DataFrame(DF_URIBE['class'].value_counts())
         df_class_uribe = df_class_uribe.reset_index()
         sns.barplot(x='index',y='class',data=df_class_uribe)
         plt.xlabel('Clase')
         plt.ylabel('Cantidad')
         plt.title('Distribución de sentimientos - Uribe')
         plt.show()
```

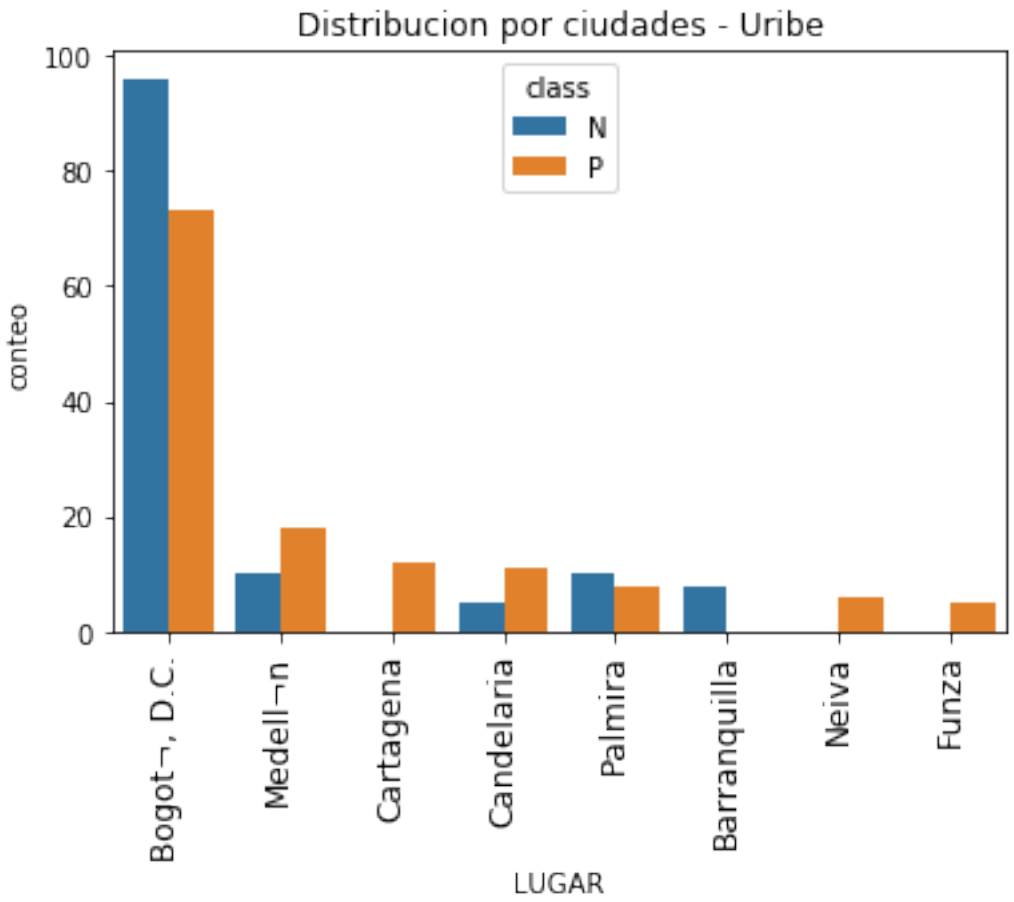


1.7.2 £Y como se distribuye en el top de ciudades por cada individio?

```
In [22]: df_lugar_p = DF_PETRO[DF_PETRO['LUGAR'].notnull()]
         df_lugar_p = df_lugar_p[['LUGAR','class']]
         df_lugar_p['conteo'] = 1
         grouped = df_lugar_p.groupby(['LUGAR','class']).sum()
         grouped = grouped.reset_index()
         grouped = grouped.nlargest(12, 'conteo')
         ax = sns.barplot(x='LUGAR',y='conteo',hue='class',data=grouped)
         ax.set_xticklabels(ax.get_xticklabels(), fontsize=12, rotation= 90)
         plt.title('Distribucion por ciudades - Petro')

Out[22]: Text(0.5, 1.0, 'Distribucion por ciudades - Petro')
```




```
In [23]: df_lugar_p = DF_URIBE[(DF_URIBE['LUGAR'].notnull()) & (DF_URIBE['LUGAR']!= 'None')]
         df_lugar_p = df_lugar_p[['LUGAR','class']]
         df_lugar_p['conteo'] = 1
         grouped = df_lugar_p.groupby(['LUGAR','class']).sum()
         grouped = grouped.reset_index()
         grouped = grouped.nlargest(12, 'conteo')
         ax = sns.barplot(x='LUGAR',y='conteo',hue='class',data=grouped)
         ax.set_xticklabels(ax.get_xticklabels(), fontsize=12, rotation= 90)
         plt.title('Distribucion por ciudades - Uribe')

Out[23]: Text(0.5, 1.0, 'Distribucion por ciudades - Uribe')
```



Para cualquiera de lso casos Bogota es por mucho, la ciudad en donde mas se tweetea de temas politicos segun estas graficas, tambien es imporatante considerar que al menos un 95% de los tweets recolecados no tenian ni coordenadas ni etiqueta de ciudad

1.8 Conclusiones

1. El score de los modelos entrenados no supero el 80% por lo que son modelos poco eficientes, esto puede ocurrir porque la data de entrenamiento fue recolectada en España y el dialecto y expresiones son muy diferentes a los usados en Colombia.

2. Sin duda los resultados muestran el gran nivel de polarizacion en el pais.

3. Segun los diagramas de nube de palabras al parecer cuando un detractor habla mal de un lider politico causa mas retweets que alguien hablando bien de este.

4. Bogota es por mucho la ciudad que mas tweetea a estos lideres politicos o al menos la que mas poblacion tiene habilitado el reconocimiento de coordendadas.

5. Es necesario crear un set de entrenamiento para el analisis de sentimientos en español y especificamente a nivel Colombia