UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Acreditación Institucional Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 8 DE MAYO–VIGENCIA 5 AÑOS

ACREDITACIÓN INSTITUCIONAL DE ALTA CALIDAD MULTICAMPUS

Vigencia por seis años

QS STARS RATED FOR EXCELLENCE

ISO 9001 icontec SC4289-1

CERTIFIED IQNet MANAGEMENT SYSTEM

UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA
VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: Systems engineer
Course: Deep Learning
Topic: sentiment analysis

Professor: Luis Fernando Castellanos Guarin
Email: Luis.castellanosg@usantoto.edu.co
Phone: 3214582098

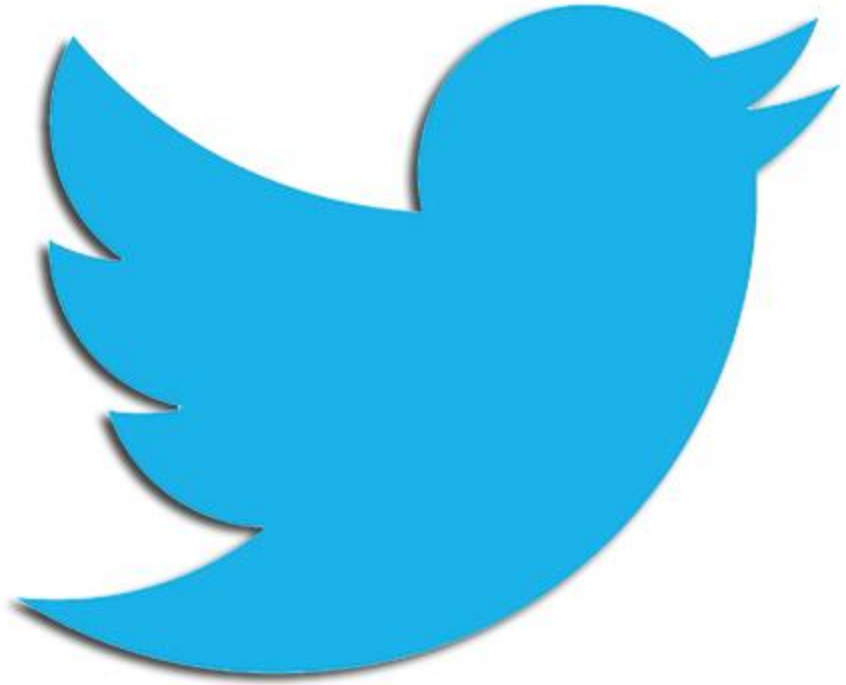
CONTENIDO

Código en Python para cada una de las fases de preparación del corpus:

- Obtención de los tres corpus.
- Convertir archivos XML del CORPUS a CSV

¡Siempre hacia lo alto!

En esta semana exploraremos el **análisis de los datos de redes sociales** a los que se accede desde **Twitter** mediante Python. Utilizaremos la API **RESTful de Twitter** para acceder a los datos sobre los usuarios de Twitter y sobre qué se tuitea.



https://www.earthdatascience.org/courses/earth-analytics-python/using-apis-natural-language-processing-twitter/get-and-use-twitter-data-in-python/

Fases de preparación del corpus

Preproceso

Tokenización

Extracción de características

Reducción de características

Ponderación

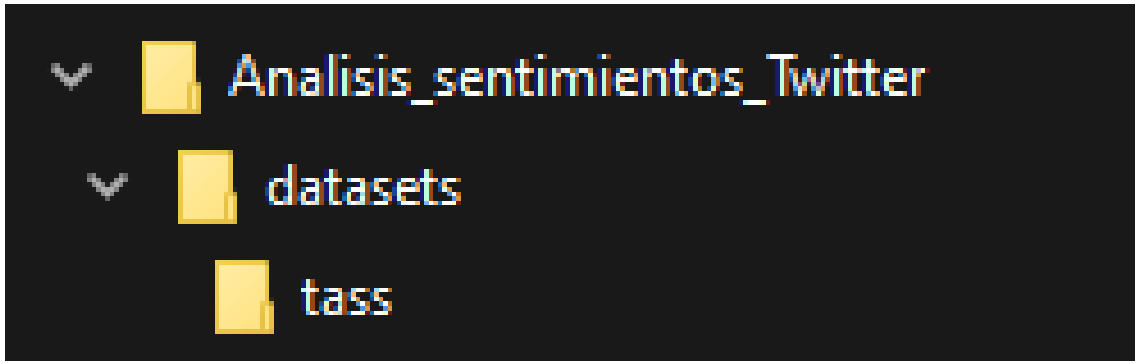
Creación del clasificador de línea base.

Proceso unico

0. Obtener corpus

Seguiremos con el mismo libro de jupyter (**ANALISIS_SENTIMIENTOS.ipynb**).

Con los archivos que descargaremos del link que nos enviaron al correo crearemos una carpeta en Google drive:



Dentro de tass dejaremos los archivos XML

0. Obtener corpus

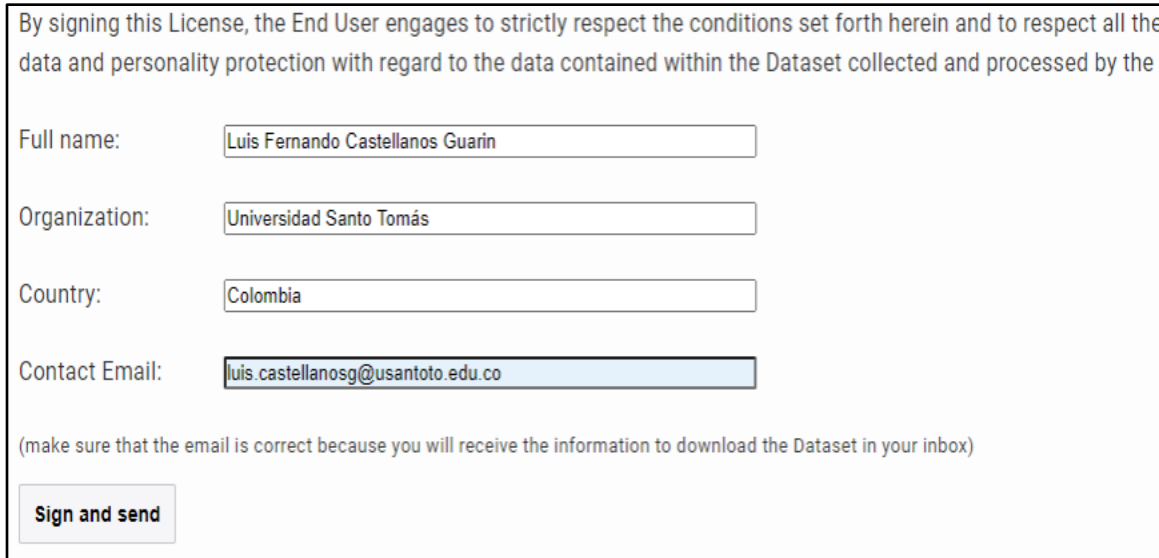
Desde el **Taller de Análisis de Sentimientos (TASS)** que pertenece a la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN) existen publicadas tres corpus que son los de más uso para nuestro idioma "El español" para analizar tweets (cada tweet esta debidamente etiquetado con el sentimiento…similar al de peliculas):

- **General Corpus**, que incluye 68000 tweets escritos en español por 150 personajes y celebridades conocidas dentro del mundo de la política, economía, comunicación y cultura y que fueron obtenidos entre noviembre de 2011 y marzo de 2012.
- **Politics Corpus (STOMPOL)**, que ofrece 2500 tweets extraídos durante la campaña de Elecciones a las Cortes Generales de España de 2011. Estos mensajes mencionan a los cuatro partidos políticos más relevantes de aquel momento: PP, PSOE, IU y UPyD.
- **International TASS Corpus (InterTASS):** contiene 3400 mensajes de Twitter escritos en español y sobre cualquier tipo de tema.

http://www.sepln.org/workshops/tass/

Luego ingresamos en la sección de DATASETS (el año más reciente) y vamos a la sección de download, registramos los datos necesarios en el formulario y esperamos que nos envíen un email con la URL de descarga.

Cree una carpeta en drive



0. Obtener corpus

Generar CSV desde XML

Cargue de librerías necesarias.

```
import xml.etree.ElementTree as etree
import csv
from os import scandir
from sklearn.model_selection import train_test_split
```

Función para convertir listas en archivos CSV

```
def list_to_csv(data, filename):
  with open(filename, 'w', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile, delimiter=',', lineterminator='\n', quoting=csv.QUOTE_NONNUMERIC)
    writer.writerows(data)
```

función para cargar de un CSV a una LISTA (messages | labels)

```
def csv_to_lists(filename):
  messages = []
  labels = []
  with open(filename, 'r', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
      messages.append(row[1])
      labels.append(row[2])
  return messages, labels
```

0. Obtener corpus

Generar CSV desde XML

Funciones para prasear xml en un lista según cada corpus

- **corpus de general tweetid | content | sentiments/polarity/value**

```
def general_tass_to_list(filename):
  tree = etree.parse(filename)
  root = tree.getroot()
  data = []

  for tweet in root:
    tweetId = tweet.find('tweetid').text
    content = tweet.find('content').text
    polarityValue = tweet.find('sentiments/polarity/value').text
    data.append([tweetId, content.replace('\n',' '), polarityValue])
  return data
```

0. Obtener corpus

Generar CSV desde XML

Funciones para prasear xml en un lista según cada corpus

- **Corpus politics tweetid | content | sentiments/polarity**

```
def politics_tass_to_list(filename):
  tree = etree.parse(filename)
  root = tree.getroot()
  data = []

  for tweet in root:
    tweetId = tweet.find('tweetid').text
    content = tweet.find('content').text
    aux = next((e for e in tweet.findall('sentiments/polarity') if e.find('entity') == None), None)
    if aux != None:
      polarityValue = aux.find('value').text
      data.append([tweetId, content.replace('\n',' '), polarityValue])
  return data
```

0. Obtener corpus

Generar CSV desde XML

Funciones para prasear xml en un lista según cada corpus

- **Corpus internacional tweetid | content | sentiments/polarity**

```
def intertass_tass_to_list(filename, qrel=None):
  tree = etree.parse(filename)
  root = tree.getroot()
  data = []

  for tweet in root:
    tweetId = tweet.find('tweetid').text
    content = tweet.find('content').text
    polarityValue = tweet.find('sentiment/polarity/value').text
    if polarityValue == None:
      polarityValue = qrel[tweetId]
      data.append([tweetId, content.replace('\n',' '), polarityValue])
  return data
```

Función para separar el 100% del corpus entre: Train : 70% - Test: 30%

```
def generate_train_test_subsets(data, size):
  codes = [d[0] for d in data]
  labels = [d[2] for d in data]
  codes_train, codes_test, labels_train, labels_test = train_test_split(codes, labels, train_size=size)
  train_data = [d for d in data if d[0] in codes_train]
  test_data = [d for d in data if d[0] in codes_test]
  return train_data, test_data
```

0. Obtener corpus

Generar CSV desde XML

Ejecutar funciones

```
#tomamos el corpus internacional (test) y generamos una lista del ID del tweet y el sentimiento para agregarlo
a la data
qrel = gold_standard_to_dict("…/Analisis_sentimientos_Twitter/datasets/tass/intertass-sentiment.qrel")
data = []
data.extend(general_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/general-test-tagged-3l.xml"))
data.extend(general_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/general-train-tagged-3l.xml"))
data.extend(intertass_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/intertass-development-
tagged.xml"))
#como el test del corpus internacional esta sin los sentimientos es necesario agregarlos : qrel
data.extend(intertass_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/intertass-test.xml", qrel))
data.extend(intertass_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/intertass-train-
tagged.xml"))
data.extend(politics_tass_to_list("…/Analisis_sentimientos_Twitter/datasets/tass/politics-test-tagged.xml"))

train, test = generate_train_test_subsets(data, size=0.7)

list_to_csv(data, '…/Analisis_sentimientos_Twitter/datasets/global_dataset.csv')
```

0. Obtener corpus

Generar CSV desde XML

Taller

Crear un archivo denominado **DatasetHelper.py,** donde se defina una clase **class DatasetHelper:**
Y dentro de ella se consoliden todas las funciones descritas en las diapositivas anteriores.

Nota:

Todos los métodos deben tener la definición **@staticmethod**

Donde se determinan que son estáticos lo que permite que en la mayoría de las funcionalidades son mejores como funciones de nivel de módulo en Python por razones de limpieza. Con una función de módulo es más fácil importar solo la función que necesita y evitar "" innecesarios, ejemplo:

```
@staticmethod
def general_tass_to_list(filename):
```

1. Preprocesamiento

Descargar librerías necesarias

```
#Descargamos la libreria de stopwords
import nltk
nltk.download('stopwords')
```

Cargamos el CSV del corpus de google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

1. Preprocesamiento

Cargar librerías necesarias y variable necesarias

```
import re                               #librería para la búsqueda y manipulación de cadenas
from nltk import TweetTokenizer         #librería para tokenizar
from nltk.stem import SnowballStemmer #algoritmo para clasificación de palabras

#variables para mejorar la escritura (opcional)
NORMALIZE = 'normalize'
REMOVE = 'remove'
MENTION = 'twmention'
HASHTAG = 'twhashtag'
URL = 'twurl'
LAUGH = 'twlaugh'
#definir que el algoritmo de clasificación use el idioma español
_stemmer = SnowballStemmer('spanish')

#definir una variable para la funcion de tokenizar (opcional)
_tokenizer = TweetTokenizer().tokenize
```

Definir listas de conversión

```
#lista de conversión para quitar las tildes a las vocales.
DIACRITICAL_VOWELS = [('á','a'), ('é','e'), ('í','i'), ('ó','o'), ('ú','u'), ('ü','u')]

#lista para corregir algunas palabras coloquiales / jerga en español (obviamente faltan más)
SLANG = [('d','de'), ('[qk]','que'), ('xo','pero'), ('xa', 'para'), ('[xp]q','porque'),('es[qk]', 'es que'),
         ('fvr','favor'),('(xfa|xf|pf|plis|pls|porfa)', 'por favor'), ('dnd','donde'), ('tb', 'también'),
         ('(tq|tk)', 'te quiero'), ('(tqm|tkm)', 'te quiero mucho'), ('x','por'), ('\+','mas')]
```

1. Preprocesamiento

Funciones necesarias

Creación de funciones que serán necesarias para el preprocesamiento de los mensajes

- **Método para normalización de risas**

```
#método para normalizar las risas
def normalize_laughs(message):
  message = re.sub(r'\b(?=\w*[j])[aeiouj]{4,}\b', LAUGH, message, flags=re.IGNORECASE)
  message = re.sub(r'\b(?=\w*[k])[aeiouk]{4,}\b', LAUGH, message, flags=re.IGNORECASE)
  message = re.sub(r'\b(juas+|lol)\b', LAUGH, message, flags=re.IGNORECASE)
  return message
```

Ejemplo:

```
print (normalize_laughs("esto muyy feliz jajajajaja o no tan feliz jejejejeje o mejor me rio a como pa pa noel JOJOJO o como en mileniams LOL"))
```

```
esto muyy feliz twlaugh o no tan feliz twlaugh o mejor me rio a como papa noel twlaugh o como en mileniams twlaugh
```

1. Preprocesamiento

Funciones necesarias

Creación de funciones que serán necesarias para el preprocesamiento de los mensajes

- **Método para eliminar o normalizar menciones, hashtags y URLS**

```
def process_twitter_features(message, twitter_features):

  message = re.sub(r'[\.\,]http','. http', message, flags=re.IGNORECASE)
  message = re.sub(r'[\.\,]#', '. #', message)
  message = re.sub(r'[\.\,]@', '. @', message)

  if twitter_features == REMOVE:
    # eliminar menciones, hashtags y URL
    message = re.sub(r'((?<=\s)|(?<=\A))(@|#)\S+', '', message)
    message = re.sub(r'\b(https?:\S+)\b', '', message, flags=re.IGNORECASE)
  elif twitter_features == NORMALIZE:
    # cuando sea necesario se normalizaran las menciones, hashtags y URL
    message = re.sub(r'((?<=\s)|(?<=\A))@\S+', MENTION, message)
    message = re.sub(r'((?<=\s)|(?<=\A))#\S+', HASHTAG, message)
    message = re.sub(r'\b(https?:\S+)\b', URL, message, flags=re.IGNORECASE)

  return message
```

Ejemplo:

```
print(process_twitter_features("Rosell, una noche. Adivina quien!! http://t.co/PPAwijRX","remove"))
```

```
Rosell, una noche. Adivina quien!!
```

1. Preprocesamiento

Funciones necesarias

Creación de funciones que serán necesarias para el preprocesamiento de los mensajes

- **Método global**

```
def preprocess(message):
  # convertir a minusculas
  message = message.lower()

  # eliminar números, retorno de linea y el tan odios retweet (de los viejos estilos de twitter)
  message = re.sub(r'(\d+|\n|\brt\b)', '', message)

  # elimar vocales con signos diacríticos (posible ambigüedad)
  for s,t in DIACRITICAL_VOWELS:
    message = re.sub(r'{0}'.format(s), t, message)

  # eliminar caracteres repetidos
  message = re.sub(r'(.)\1{2,}', r'\1\1', message)

  # normalizar las risas
  message = normalize_laughs(message)

  # traducir la jerga y terminos coloquiales sobre todo en el español
  for s,t in SLANG:
    message = re.sub(r'\b{0}\b'.format(s), t, message)

  #normalizar/eliminar hashtags, menciones y URL
  message = process_twitter_features(message, _twitter_features)

  #Convertir las palabras a su raiz ( Bonita, bonito) -> bonit
  if _stemming:
    message = ' '.join(_stemmer.stem(w) for w in _tokenizer(message))

  return message
```

1. Preprocesamiento

Funciones necesarias

Taller:

Usando las funciones:

- Para cargar de un CSV a una LISTA, `def csv_to_lists(filename)`
- Preprocesamiento de textos, `def preprocess(message)`

Crear el Código en python para cargar el archivo **CSV** *"datasets/train_dataset_30.csv"* y realizar el preprocesamiento a cada uno de los tweets y salvar nuevamente el dataset en un nuevo csv denominado **train_dataset_30_clean.csv**

1. Preprocesamiento

Funciones necesarias

Aplicamos preprocesamiento al CSV y creamos un nuevo CSV limpio

```
import numpy as np
import pandas as pd

df = pd.read_csv('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/datasets/dataset_2017_full.csv', encodin
g='utf-8')
#asignamos nombres a las columnas del csv para facilitar la busqueda de información
df.columns = ['tweetid', 'tweet','sentiment']
#aplicamos el preprocesamiento a los tweets con steaming =false
df['tweet'] = df['tweet'].apply(preprocess)
#eliminamos la columna tweetid que no nos sirve para entrenar y si nos genera mas uso de memoria
df = df.drop(columns="tweetid")
#Es mejor trabajar con valores enteros que con letras
#por lo tanto reemplazaremos los sentimientos que estan como NONE->-1 | NEU -> 0 | P->1 | N->2
df.loc[df['sentiment'] == 'NONE', 'sentiment'] = '-1'
df.loc[df['sentiment'] == 'NEU', 'sentiment'] = '0'
df.loc[df['sentiment'] == 'P', 'sentiment'] = '1'
df.loc[df['sentiment'] == 'N', 'sentiment'] = '2'
df["sentiment"].unique()
#guardamos el dataset en un nuevo CSV para facilitar su posterior uso
df.to_csv('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/datasets/dataset_2017_full_clean.csv', index=Fa
lse, encoding='utf-8')
```

5.2. tokenización

Función de tokenizar

Esta función permite convertir cada uno de los tweets en un vector donde cada uno de los elementos es una palabra o símbolo gramatical.

```
def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) +' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized
```

5.3. Extracción de características

```
#p2.2: función para extraer un documento del dataset
print("p2.2: funcion para extraer un documento del dataset  ")
def stream_docs(path):
    with open(path, 'r', encoding='utf-8') as csv:
        next(csv)  # skip header
        for line in csv:
            text, label = line[:-3],  int(line[-2])
            yield text, label
#p2.3: función que tomara una secuencia de documentos y devolverá un número particular de documentos
def get_minibatch(doc_stream, size):
    docs, y = [], []
    try:
        for _ in range(size):
            text, label = next(doc_stream)
            docs.append(text)
            y.append(label)
    except StopIteration:
        return None, None
    return docs, y
```

5.4. Entrenamos el modelo

usaremos regresión logística porque es menos costoso en tiempo de procesamiento que “support vector machine”

```
path='/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/datasets/dataset_2017_full_clean.csv'
#p2: definimos una versión liviana de CountVectorizer+TfidfVectorizer llamada HashingVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.linear_model import SGDClassifier

vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)

#definimos como algoritmo la regressión logistica en el decenso gradiante

clf = SGDClassifier(loss='log', random_state=1, max_iter=1)
doc_stream = stream_docs(path)
#p3. entrenamos
import re
import numpy as np
import pyprind
from nltk.corpus import stopwords
stop = stopwords.words('spanish')
pbar = pyprind.ProgBar(50)
#definimos las clases con las cuales vamos a entrenar
classes = np.array([-1,0, 1,2])
#hacemos 50 repeticiones
for _ in range(50):
  #tomaremos grupos de 500 tweets para entrenar
    X_train, y_train = get_minibatch(doc_stream, size=500)
    if not X_train:
        break
    X_train = vect.transform(X_train)
    clf.partial_fit(X_train, y_train, classes=classes)
    pbar.update()
#probamos la eficiencia del modelo con 500 tweets .
X_test, y_test = get_minibatch(doc_stream, size=500)
X_test = vect.transform(X_test)
print('Presición del modelo: %.3f' % clf.score(X_test, y_test))
#recalibramos el modelo.
clf = clf.partial_fit(X_test, y_test)
```

…Creamos una IA que clasifique (positivo, negativo, neutro) un texto
con una acertabilidad del 80%...

con eso creamos SKYNET…QUE SUSTOOOOOOOOOO

¡Siempre hacia lo alto!

Serializamos (congelamos) el modelo

Serializamos (congelamos) el modelo para usarlo fuera de google colaboratory, de esta forma crearemos dos archivos portables que se podrían usar en Android/IOS/Windows/Linux o en una pagina web.

```
import pickle
import os
#creo una carpeta en mi google drive para guardar los archivos serializados
dest = os.path.join('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/twitterclassifier',
'pkl_objects')
if not os.path.exists(dest):
    os.makedirs(dest)
#convertimos el clasificador y el stopword en archivo/objectos pkl
pickle.dump(stop, open(os.path.join(dest, 'stopwords.pkl'), 'wb'), protocol=4)
pickle.dump(clf, open(os.path.join(dest, 'classifier.pkl'), 'wb'), protocol=4)
#Es importante recordar que deben verificar que los dos archivos esten en su drive
```

Serializamos (congelamos) el modelo

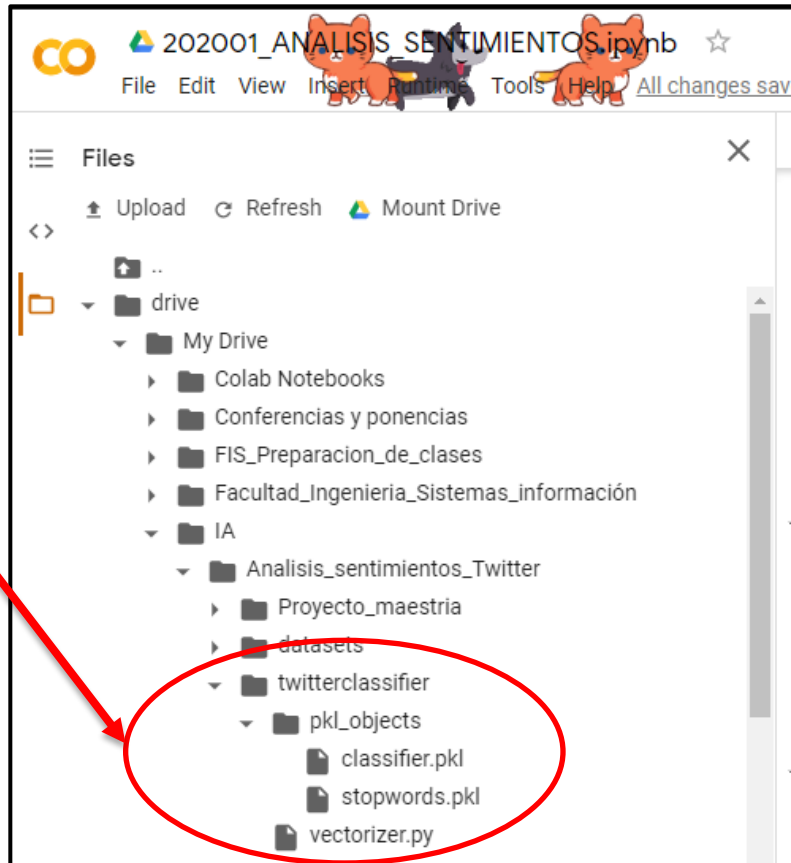
guardamos en drive en la carpeta de **twitterclassifier** un archivo de Python llamado **vectorizer.py** para usar el **HashingVectorizer** más tarde (me permitirá convertir cualquier texto en un vector).

```
from sklearn.feature_extraction.text import HashingVectorizer
import re
import os
import pickle

cur_dir = os.path.dirname(__file__)
stop = pickle.load(open(
                os.path.join(cur_dir,
                'pkl_objects',
                'stopwords.pkl'), 'rb'))

def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',
                           text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) \
                + ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)
```



Probemos a ver si funciona

Cambiamos la basepath (directorio por defecto) de Python a la carpeta de **Twitterclassifier**

```
import os
os.chdir('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/twitterclassifier')
```

Deserializamos los estimadores

```
import pickle
import re
import os
from vectorizer import vect
clf = pickle.load(open(os.path.join('pkl_objects', 'classifier.pkl'), 'rb'))
```

Probemos a ver si funciona

Clasifiquemos un texto

```
import numpy as np
#NONE->-1 | NEU -> 0 | P->1 | N->2
label = {-1:'Sin sentimiento', 0:'Neutro', 1:'Positivo',2: 'Negativo'}

#example = ['Te odio más que a la muerte']
example1 = 'covid19 te ODIOOOO'
example = [example1]
#convertimos el texto en un vector de palabras y extraemos sus caracteristicas https://scikit-
learn.org/stable/modules/feature_extraction.html
textConvert = vect.transform(example)
print('*Predicción: %s\n*Probabilidad: %.2f%%'%(label[clf.predict(textConvert)[0]], np.max(clf.predict_proba(textCon
vert))*100))
print('*Predicción: %s'%label[clf.predict(textConvert)[0]])
print(np.max(clf.predict_proba(textConvert))*100)
```

RECORREMOS LOS TWEETS DESCARGADOS Y LOS CLASIFICAMOS

Cargaremos los tweets de COVID19 (aproximadamente 1000)

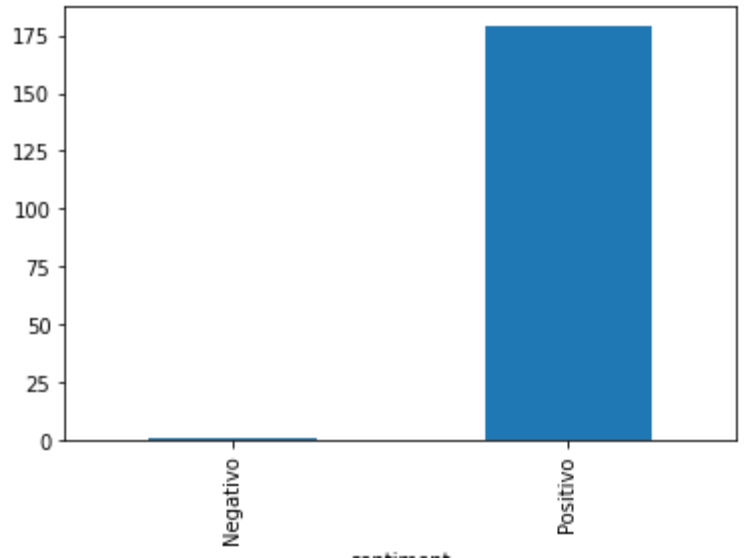
```
import numpy as np
import pandas as pd
import pyprind

df = pd.read_csv('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/datasets/TWEETS_LGGG_COVID19.csv', enco
ding='utf-8')
#creamos una columna llamada Sentimient donde guardaremos la predicción
df['sentiment'] =''
#creamos una columna llamada Probability donde guardaremos la acertabilidad que dio el clasificador
df['probability']=0
#conversión de sentimientos (numeros a palabras)= NONE->-1 | NEU -> 0 | P->1 | N->2
label = {-1:'Sin sentimiento', 0:'Neutro', 1:'Positivo',2: 'Negativo'}
for rowid in range(len(df.index)):
  text=df['text'][rowid]
  textConvert = vect.transform([text])
  df['sentiment'][rowid]=label[clf.predict(textConvert)[0]]
  df['probability'][rowid]=np.max(clf.predict_proba(textConvert))*100
  pbar.update()
df.head(20)
#df.to_csv('/content/drive/My Drive/IA/Analisis_sentimientos_Twitter/datasets/TWEETS_LGGG_COVID19_analysis.csv', i
ndex=False, encoding='utf-8')
```

Generemos gráficos estadísticos

Generemos algunos gráficos estadísticos que nos ayuden a verificar que tan eficiente fue el modelo

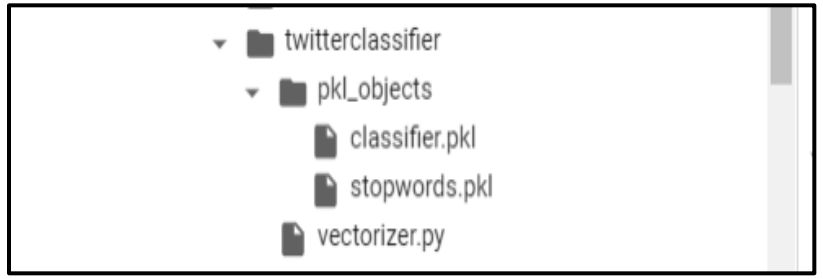
```
import matplotlib.pyplot as plt
#sentimientos = df["sentiment"].unique()
df.groupby('sentiment')['location'].nunique().plot(kind='bar')
print(df.groupby(['sentiment']).size())
#df.groupby(['sentiment']).size().unstack().plot(kind='bar',stacked=True)
plt.show()
```



Taller investigativo

Taller: Crear un repositorio en **Github** denominado **DeepLearning** y en el una carpeta denominada **Analisis_sentimientos_Twitter**.
Dentro de esta carpeta debe estar:

- Carpeta del clasificador



- Libro de jupyter (código fuente)
- Csv con los tweets para hacerles análisis y de resultado (sentimiento).
- **Pdf** con los printscreen de los gráficos estadísticos.

Taller investigativo

Taller: Desde el CRAI-de la USTA TUNJA (https://ustatunja.edu.co/inicio-biblioteca)

Investigar que cuanto material académico:

- Libros
- Capítulos de libro
- Artículos
- Ponencias

Se tienen sobre análisis de sentimientos usados en Twitter.

Adicional a esto tome un **articulo**, el de su preferencia y realice el siguiente análisis:

- Cual fue el corpus que usaron (de donde obtuvieron la base o si ellos la crearon).
- Que técnica de aprendizaje utilizaron para entrenar el modelo.
- Que acertabilidad (probabilidad de acertar) obtuvieron (¿si fue superior al 80%?).
- A que conclusiones llegaron los investigadores.
- Responda si con lo que ha aprendido en clase de Deep learning usted recomendaría ese proyecto?