

#### Máster en Big Data

## Asignatura: Casos de analítica

## 

Apellidos: \_Ripoll Batlló\_\_\_\_\_\_\_\_\_\_ Nombre: \_Albert\_\_ Curso: **2023/24**

**Caso 2: Sentimientos en Twitter**

Twitter es una red social que ha degradado a una comunidad llena de Trolls (trollear es la acción de hacer sentir mal o hacer enojar a alguien con bromas pesadas o comentarios fuera de lugar) o Cuñados (aquellos que comentan sobre cualquier asunto, queriendo aparentar ser más listos que los demás). El dicho “No alimentes al Troll” no se aplica en Twitter y estos se retroalimentan hasta el punto de conseguir que conversaciones de alto nivel se conviertan en un patio de escuela. Entre Trolls y Cuñados Twitter ya no es lo que era.

Gracias a las técnicas de análisis de texto (*Text Analysis*) podemos combatir los Trolls y Cuñados y devolver Twitter a sus años de esplendor. *Text Analysis* consiste en extraer información a partir de datos de lenguaje humano para comprender cómo otros seres humanos entienden el mundo, agruparlos y extraer patrones de comportamiento.

En este Caso 2 debes realizar un análisis sentimental, sintáctico y gramatical de comentarios Twitter. La base de datos la puedes descargar desde eStudy (Caso 2 dataset), la cual contiene un CSV de mensajes enviados a Twitter con las siguientes columnas:

1. Puntuación sentimental o polaridad (-5 = negativa … 0 = neutral … 5 = positiva) (por calcular)
2. Id del tweet
3. Fecha del tweet (Sat May 16 23:58:44 UTC 2009)
4. Búsqueda. Si no hay búsqueda, el valor es NO\_QUERY
5. Usuario que ha tweeteado
6. Texto del tweet

Con estos datos se os propone que apliquéis técnicas analíticas y de visualización para responder a las siguientes preguntas. No hay restricciones acerca de las técnicas ni tecnologías a utilizar siempre y cuando los resultados sean reproducibles y estén debidamente justificados. No obstante, las siguientes librerías y códigos de ejemplo os pueden ser muy útiles para responderlas:

|  |
| --- |
| **Librería NLTK**  <https://www.nltk.org/install.html>  Propósito: Trabajar con datos en lenguaje humano. |

|  |
| --- |
| **Librería texstat**  <https://pypi.org/project/textstat/>  Propósito: Calcular estadística a partir de datos en lenguaje humano. |

|  |
| --- |
| **Unsupervised-Text-Clustering using Natural Language Processing (NLP)**  Para realizar un conglomerado analítico de un corpus documental/textos se acostumbra a seguir los siguientes pasos genéricos. La técnica consiste en crear un vector cuantitativo a partir de los textos, previa limpieza y transformación, para aplicar técnicas de conglomerado:   1. Eliminar caracteres de puntuación, espacios adicionales, dígitos u otros caracteres que puedan entorpecer el análisis textual 2. Tokenizar y eliminar *Stopwords*. Se requiere un diccionario de palabras para quitar aquellas que puedan entorpecer el análisis textual. Por ejemplo, se puede utilizar “*from nltk.corpus import stopwords*”. Ejemplo: [NLTK stop words - Python Tutorial (pythonspot.com)](https://pythonspot.com/nltk-stop-words/) 3. Encontrar la raíz de las palabras aplicando *lemmatization* o *stemming*. 4. Aplicar vectorizado del tokenizado para calcular apariciones de los tokens y cuantificar los tweets. Se pueden usar distintos cálculos, por ejemplo Bag-of-Words, Word2Vec, o TFIDF con “*from sklearn.feature\_extraction.text import TfidfVectorizer*” 5. Aplicar clustering con técnicas adecuadas. Por ejemplo, Kmeans previo cálculo del número de clusters con técnicas como Elbow. |

|  |
| --- |
| **Transformers**  Huggingface pone a disposición una manera muy asequible de realizar análisis sentimentales con modelos pre-entrenados. Sigue estos dos enlaces para poder realizar las preguntas extras del caso:   1. [Getting Started with Sentiment Analysis using Python](https://huggingface.co/blog/sentiment-analysis-python) 2. [Pipelines](https://huggingface.co/docs/transformers/main_classes/pipelines) |

**ANÁLISIS para la pregunta 1**

1. Cuál es la distribución de las polaridades y complejidad de lectura/escritura de los tweets en el dataset?
   1. ¿Hay una mayor cantidad de tweets positivos, negativos o neutrales?
   2. ¿Cómo se relacionan las distintas polaridades según la complejidad de lectura/escritura de los tweets?

Se cargan los datos en un dataframe.

|  |
| --- |
| df = pd.read\_csv(ruta\_base + 'training.1600000.processed.noemoticon.csv')  df.columns = ['points', 'id', 'date', 'search', 'user', 'text']  df.head(2) |

En primera instancia, pensamos que los nombres de los usuarios a los que se responden los mensajes no sería una parte del texto a analizar ya que no aporta significado a lo que la persona está redactando. Así que lo quitamos. Más tarde nos dimos cuenta que podríamos usar esta información como estimación del número de seguidores en la pregunta 3. Es decir, cada contestación a un usuario implica un seguidor.

|  |
| --- |
| # Si la primera palabra de text empieza por @, eliminar toda la palabra  def eliminar\_primera\_palabra(text):      if text[0] == '@':          return re.sub(r'^@\w+\s', '', text)      else:          return text  df['text'] = df['text'].apply(eliminar\_primera\_palabra) |

Con el objetivo de que el texto sea más fácil de analizar, se limpia el texto eliminando todo carácter que no sea letra, número o espacio (**[^\w\s]**), se borran los dígitos (**\d)** y se borran los espacios adicionales, quedándonos con uno solo (**\s+**). Finalmente se convierte todo el texto en minúscula (.lower()). Si no eliminásemos los espacios en blanco extras, habría problemas en detectar palabras como el conjunto de caracteres que se separan de los espacios. Los signos de puntuación y dígitos entorpecerían el análisis ya que se juntarían con las palabras. Las mayúsculas impedirían que ciertas palabras fuesen reconocidas. Por ese motivo, se limpia de estos elementos el texto.

|  |
| --- |
| def clean\_text(text):      # Elimina caracteres de puntuación y dígitos      text = re.sub(r'[^\w\s]', '', text)      text = re.sub(r'\d', '', text)      # Elimina espacios adicionales      text = re.sub(r'\s+', ' ', text)      # Convierte el texto a minúsculas      text = text.lower()      return text  df['cleaned\_text'] = df['text'].apply(clean\_text) |

Las “Stop words” como ‘the’, ‘and’ y ‘I’ son palabras comúnmente usadas en inglés que normalmente no aporta información importante sobre el tema de un texto. Eliminándo estas palabras, se puede identificar más fácilmente los términos únicos y relevantes. Fuente: <https://pythonspot.com/nltk-stop-words/>.

|  |
| --- |
| import nltk  nltk.download('stopwords')  nltk.download('punkt')  nltk.download('wordnet')  from nltk.corpus import stopwords  from nltk.tokenize import sent\_tokenize, word\_tokenize  stops = set(stopwords.words('english'))  print(stops)  def tokenize\_text(text):    words = word\_tokenize(text)    wordsFiltered = []    for word in words:      if word not in stops:        wordsFiltered.append(word)    # wordsFiltered = [word for word in words if word not in stops]    return ' '.join(wordsFiltered)    df['tokenized\_text'] = df['cleaned\_text'].apply(tokenize\_text) |

En términos lingüísticos las palabras no son más que transformaciones derivadas de la raíz, lema o lexema de la palabra, que sería la base de la palabra que aporta todo su significado. Las conjugaciones o derivaciones de las palabras pueden ser muy numéricas. Para analizar correctamente el significado de un texto es conveniente sacar la raíz, lema o lexema de las palabras y usar ese para el análisis de significado del texto. Existen dos transformaciones posibles para lograr la raíz, lema o lexema de la palabra:

• Lematización (Lematize):

* La lematización utiliza el conocimiento de la morfología y la estructura gramatical de las palabras para encontrar su lexema, lema o raíz.
* Se basa en reglas lingüísticas y utiliza un vocabulario léxico (diccionario) para relacionar las palabras con sus formas base.
* La lematización puede producir formas de palabras que son válidas y legibles, ya que tiene en cuenta la morfología y la función gramatical de las palabras.
* Es más lenta y computacionalmente más intensiva que la derivación, ya que implica un análisis más profundo de las palabras.

• Derivación (Stemming):

* La derivación elimina sufijos y prefijos, sin tener en cuenta la estructura gramatical para encontra el lexema, lema o raíz.
* Es un proceso más simplificado y más rápido que la lematización, ya que se basa en reglas heurísticas simples sin conocimiento lingüístico profundo.
* Puede producir formas de palabras que no son válidas o legibles, ya que no tiene en cuenta la morfología o la función gramatical.
* Debido a su simplicidad, la derivación puede no ofrecer resultados tan precisos como la lematización, pero puede ser adecuada en situaciones donde la velocidad y la simplicidad son más importantes que la precisión absoluta.

|  |
| --- |
| from nltk.stem import WordNetLemmatizer, PorterStemmer  lemmatizer = WordNetLemmatizer()  stemmer = PorterStemmer()  def lemmatize\_words(text):    words = word\_tokenize(text)    lemmatized\_words  = []    for word in words:      lemmatized\_words.append(lemmatizer.lemmatize(word))    # lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]    return ' '.join(lemmatized\_words)  def stem\_words(text):    words = word\_tokenize(text)    stemmed\_words  = []    for word in words:      stemmed\_words.append(stemmer.stem(word))    # stemmed\_words = [stemmer.stem(word) for word in words]    return ' '.join(stemmed\_words)  df['lematized\_text'] = df['tokenized\_text'].apply(lemmatize\_words)  df['stemmed\_text'] = df['tokenized\_text'].apply(stem\_words) |

Una vez limpiado, tokenizado, lematizado y stemmatizado el texto, eliminamos la columna original “texto” ya que hemos descubierto al guardar el dataframe con esta columna se generan problemas debido a la presencia de signos de puntuación como , o ; presentes en “texto”.

Después, guardamos el dataframe para volver a usarlo más tarde sin tener que volver a ejecutar el código que tarda varios minutos (sobretodo el lematize\_words y stem\_words).

|  |
| --- |
| df.drop(columns=['text'], inplace=True)  # Save df  df.to\_csv(os.path.join(ruta\_base, "stemmed\_lemmatized01.csv"), index=False)  # Read df  df = pd.read\_csv(ruta\_base + "stemmed\_lemmatized01.csv")  df.tail(2) |

Comprobamos si los textos generados ‘lematized\_text’ se han generado sin ningún NaN y vemos que hay 6531 textos con algún valor NaN. Eliminamos esas filas con NaN. Comprovamos que aún no hayan quedado NaN en ‘stemmed\_text’.

|  |
| --- |
| df['lematized\_text'].isnull().sum()    df = df.dropna(subset=['lematized\_text'])  df['stemmed\_text'].isnull().sum() |

Para responder a la primera pregunta, miramos como están distribuidos los puntos y observamos que hay 800.000 valores en cada categoría de puntos (0→negativo, 4→positivo). Esto respondería a la pregunta 1a indicando que igual de hay igual de valores positivos que negativos.

|  |
| --- |
| df['points'].value\_counts() |

Para tener una puntuación de polaridad más ajustada, ajustada de -5 a 5 como pide el enunciado, usamos la función SentimentIntensityAnalyzer de la librería nltk.stentiment. Esta función devuelve un análisis de -1 a 1 con decimales sobre el sentimiento. Por eso lo multiplicamos por 5 y lo redondeamos a entero, para que esté correctamente rangeado con los valores -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5. Se analiza por separado los puntos del texto lematizado y stematizado, para poder hacer futuras comparaciones.

|  |
| --- |
| import nltk  from nltk.sentiment import SentimentIntensityAnalyzer  nltk.download('vader\_lexicon')  sia = SentimentIntensityAnalyzer()  # Creamos colmna Points de -5 a 5 redondeado a entero  df['points\_lem'] = df['lematized\_text'].apply(lambda x: round(sia.polarity\_scores(x)['compound'] \* 5))  df['points\_stem'] = df['stemmed\_text'].apply(lambda x: round(sia.polarity\_scores(x)['compound'] \* 5))  df.tail(2) |

Para responder a la pregunta 1a, contamos los valores de points\_lem y points\_stem. Consideramos negativo si el valor es -5, -4, -3, -2 o -1. Consideramos 0 como neutro. Y consideramos positivo si el valor es 1, 2, 3, 4 o 5.

|  |
| --- |
| negativos = (df['points\_lem'] < 0).sum()  cero = (df['points\_lem'] == 0).sum()  positivos = (df['points\_lem'] > 0).sum()  etiquetas = ['Negativos', 'Cero', 'Positivos']  datos = [negativos, cero, positivos]  colores = ['red', 'grey', 'green']  # Crear Pie Chart  plt.figure(figsize=(4, 3))  plt.pie(datos, labels=etiquetas, colors=colores, autopct=lambda p: '{:.1f}%\n({:d})'.format(p, int(p \* sum(datos) / 100)), startangle=140)  plt.title('Tweets negativos, neutrales y positivos en points\_lem')  plt.axis('equal')  # Hace que el gráfico sea circular  plt.show() |

Para poder analizar la complejidad de escritura, nos basamos en dos métricas:

• La longitud del texto. Entendemos que cuanto mayor es este valor, más complejo es el texto.

• La diversidad léxica del texto. Esta métrica es el tanto por 1 de palabras únicas respecto a las palabras redactadas. Por ejemplo 1 significaría que el 100% de las palabras usadas son únicas. El usuario usa léxico distinto. 0,5 significaría que el 50% de las palabras son distintas.

|  |
| --- |
| import nltk  nltk.download('punkt')  nltk.download('averaged\_perceptron\_tagger')  def calcular\_longitud(texto):      palabras = nltk.word\_tokenize(texto)      return len(palabras)  def calcular\_diversidad\_lexica(texto):      palabras = nltk.word\_tokenize(texto)      palabras\_unicas = set(palabras)      if len(palabras) == 0:        diversidad\_lexica = 0      else:        diversidad\_lexica = len(palabras\_unicas) / len(palabras)      return diversidad\_lexica  df['diversity\_lem'] = df['stemmed\_text'].apply(calcular\_diversidad\_lexica)  df['diversity\_stem'] = df['lematized\_text'].apply(calcular\_diversidad\_lexica)  df['longitud\_lem'] = df['lematized\_text'].apply(calcular\_longitud)  df['longitud\_stem'] = df['stemmed\_text'].apply(calcular\_longitud) |

Para responder a la pregunta 1b miramos las correlaciones que hay entre las métricas de puntos ('points\_lem' y 'points\_stem') con las métricas de complejidad ('diversity’ y 'diversity\_stem'). Al calcular los coeficientes vemos que estos son muy cercanos a 0, lo que indica que no hay correlación. Los gráficos también indican una correlación nula. Los resultados concluyen que no hay correlación entre la polaridad y la complejidad del Tweet.

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  # Calcular los coeficientes de correlación  correlation\_lem\_diversity\_points = df['diversity\_lem'].corr(df['points\_lem'])  correlation\_stem\_diversity\_points = df['diversity\_stem'].corr(df['points\_stem'])  correlation\_lem\_longitud\_points = df['longitud\_lem'].corr(df['points\_lem'])  correlation\_stem\_longitud\_points = df['longitud\_stem'].corr(df['points\_stem'])  # Imprimir los coeficientes de correlación  print("Coeficiente de correlación entre diversity\_lem y points\_lem:", correlation\_lem\_diversity\_points)  print("Coeficiente de correlación entre diversity\_stem y points\_stem:", correlation\_stem\_diversity\_points)  print("Coeficiente de correlación entre longitud\_lem y points\_lem:", correlation\_lem\_longitud\_points)  print("Coeficiente de correlación entre longitud\_stem y points\_stem:", correlation\_stem\_longitud\_points)  # Crear gráficos de dispersión  fig, axs = plt.subplots(2, 2, figsize=(12, 8))  # Gráfico 1: diversity\_lem vs points\_lem  axs[0, 0].scatter(df['diversity\_lem'], df['points\_lem'])  axs[0, 0].set\_title('diversity\_lem vs points\_lem')  axs[0, 0].set\_xlabel('diversity\_lem')  axs[0, 0].set\_ylabel('points\_lem')  # Gráfico 2: diversity\_stem vs points\_stem  axs[0, 1].scatter(df['diversity\_stem'], df['points\_stem'])  axs[0, 1].set\_title('diversity\_stem vs points\_stem')  axs[0, 1].set\_xlabel('diversity\_stem')  axs[0, 1].set\_ylabel('points\_stem')  # Gráfico 3: longitud\_lem vs points\_lem  axs[1, 0].scatter(df['longitud\_lem'], df['points\_lem'])  axs[1, 0].set\_title('longitud\_lem vs points\_lem')  axs[1, 0].set\_xlabel('longitud\_lem')  axs[1, 0].set\_ylabel('points\_lem')  # Gráfico 4: longitud\_stem vs points\_stem  axs[1, 1].scatter(df['longitud\_stem'], df['points\_stem'])  axs[1, 1].set\_title('longitud\_stem vs points\_stem')  axs[1, 1].set\_xlabel('longitud\_stem')  axs[1, 1].set\_ylabel('points\_stem')  # Ajustar la disposición de los gráficos  plt.tight\_layout()  # Mostrar los gráficos  plt.show()  Coeficiente de correlación entre diversity\_lem y points\_lem: -0.005305037096272493  Coeficiente de correlación entre diversity\_stem y points\_stem: -0.006265023303434058  Coeficiente de correlación entre longitud\_lem y points\_lem: 0.11683805265841983  Coeficiente de correlación entre longitud\_stem y points\_stem: 0.11593136542678646 |

**ANÁLISIS para la pregunta 2**

1. ¿Existen patrones gramaticales o sintácticos comunes en los tweets con polaridad positiva o negativa? Por ejemplo, puede que los tweets positivos tiendan a utilizar más palabras de agradecimiento o elogios, mientras que los tweets negativos utilizan más palabras de crítica o enojo.

En el final del análisis 1 vimos que no hay correlación entre polaridad positiva y negativa y complejidad del Tweet. No obstante vamos a ver qué palabras se usan en Tweets positivos y negativos para ver si hay algún conjunto de palabras que se asocien a un tipo u otro de polaridad. Se destaca que:

* En los Tweets positivos entre las 10 palabras más usadas hay: good, love, like, lol, thanks, well.
* En los Tweets negativos entre las 10 palabras más usadas hay: sad, dont, bad, hate, ill.

|  |
| --- |
| import pandas as pd  import nltk  from nltk.tokenize import word\_tokenize  from nltk.corpus import stopwords  from collections import Counter  # Descargar los recursos necesarios de NLTK (puedes omitir este paso si ya los tienes descargados)  nltk.download('punkt')  nltk.download('stopwords')  # Tokenización y limpieza de texto  def limpiar\_texto(texto):      palabras = word\_tokenize(texto)      listado = [token for token in palabras]      return listado  # Agrupar los tweets por polaridad  tweets\_positivos\_lem = ' '.join(df[df['points\_lem'] > 0]['lematized\_text'])  tweets\_negativos\_lem = ' '.join(df[df['points\_lem'] < 0]['lematized\_text'])  # Tokenización y limpieza de texto para tweets positivos y negativos  tokens\_positivos\_lem = limpiar\_texto(tweets\_positivos\_lem)  tokens\_negativos\_lem = limpiar\_texto(tweets\_negativos\_lem)  # Análisis de frecuencia de palabras  frecuencia\_positivos\_lem = Counter(tokens\_positivos\_lem)  frecuencia\_negativos\_lem = Counter(tokens\_negativos\_lem)  # Identificación de las palabras más frecuentes en cada grupo  palabras\_comunes\_positivas\_lem = frecuencia\_positivos\_lem.most\_common(10)  palabras\_comunes\_negativas\_lem = frecuencia\_negativos\_lem.most\_common(10)  # Imprimir los resultados  print("Palabras más comunes en tweets positivos:", palabras\_comunes\_positivas\_lem)  print("Palabras más comunes en tweets negativos:", palabras\_comunes\_negativas\_lem)  Palabras más comunes en tweets positivos: [('im', 84217), ('good', 78634), ('love', 63373), ('day', 58358), ('like', 58296), ('lol', 46510), ('get', 42069), ('u', 39497), ('thanks', 37066), ('well', 36034)]  Palabras más comunes en tweets negativos: [('im', 54253), ('miss', 21745), ('sad', 21201), ('day', 20992), ('dont', 20488), ('get', 20443), ('bad', 19989), ('go', 18452), ('hate', 18300), ('ill', 17778)] |

**ANÁLISIS para la pregunta 3**

1. ¿Qué usuarios tienden a generar tweets con una polaridad más positiva o negativa? ¿Hay alguna relación entre la polaridad de los tweets y el número de seguidores de un usuario?

Dado que en nuestro data frame borramos las palabras seguidas de @, necesitamos recuperar esta información para considerar la cantidad de contestaciones como el número de seguidores. El plan es el siguiente, cargar solo la columna “texto” de los datos originales, coger la primer palabra que sigue del @, contar la cantidad de palabras únicas, unir por nombre de usuario la cantidad de esas palabras simulando la cantidad de seguidores.

1. ¿Hay alguna palabra o conjunto de palabras específicas que estén asociadas con tweets de polaridad extrema?
   1. ¿Estas palabras son más comunes en tweets sobre un tema en particular o están distribuidas en todo el dataset?

Analizamos el porcentaje de textos que contienen las palabras positivas o negativas en el dataframe y lo comparamos con lo mismo para los textos que contienen la palabra Obama (o sea, son de este tema).

Vemos que la cantidad positiva es similar en ambos casos, pero en el caso de que se habla de Obama, hay algo mayor de palabras negativas. Este es el análisis:

|  |
| --- |
| # Definir las palabras específicas a contar  palabras\_positivas = ['good', 'love', 'like', 'lol', 'thanks', 'well']  palabras\_negativas = ['sad', 'dont', 'bad', 'hate', 'ill']  # Inicializar contadores  contador\_obama\_positivo = 0  contador\_df\_positivo = 0  contador\_obama\_negativo = 0  contador\_df\_negativo = 0  # Iterar sobre cada texto en obama['lematized\_text'] y df['lematized\_text']  for texto in obama['lematized\_text']:      if any(palabra in texto for palabra in palabras\_positivas):          contador\_obama\_positivo += 1      if any(palabra in texto for palabra in palabras\_negativas):          contador\_obama\_negativo += 1  for texto in df['lematized\_text']:      if any(palabra in texto for palabra in palabras\_positivas):          contador\_df\_positivo += 1      if any(palabra in texto for palabra in palabras\_negativas):          contador\_df\_negativo += 1  # Calcular el porcentaje de textos que contienen al menos una palabra positiva o negativa en cada conjunto  total\_textos\_obama = len(obama)  total\_textos\_df = len(df)  porcentaje\_obama\_positivo = (contador\_obama\_positivo / total\_textos\_obama) \* 100  porcentaje\_df\_positivo = (contador\_df\_positivo / total\_textos\_df) \* 100  porcentaje\_obama\_negativo = (contador\_obama\_negativo / total\_textos\_obama) \* 100  porcentaje\_df\_negativo = (contador\_df\_negativo / total\_textos\_df) \* 100  # Mostrar los resultados  print("Porcentaje de textos con palabras positivas en obama['lematized\_text']:", "{:.2f}%".format(porcentaje\_obama\_positivo))  print("Porcentaje de textos con palabras positivas en df['lematized\_text']:", "{:.2f}%".format(porcentaje\_df\_positivo))  print("Porcentaje de textos con palabras negativas en obama['lematized\_text']:", "{:.2f}%".format(porcentaje\_obama\_negativo))  print("Porcentaje de textos con palabras negativas en df['lematized\_text']:", "{:.2f}%".format(porcentaje\_df\_negativo))  # Crear piechart para palabras positivas  plt.figure(figsize=(10, 5))  plt.subplot(1, 2, 1)  plt.pie([porcentaje\_obama\_positivo, porcentaje\_df\_positivo], labels=['Obama', 'DF'], autopct='%1.2f%%', startangle=140, colors=['green', 'lightgreen'])  plt.title('Palabras Positivas')  # Crear piechart para palabras negativas  plt.subplot(1, 2, 2)  plt.pie([porcentaje\_obama\_negativo, porcentaje\_df\_negativo], labels=['Obama', 'DF'], autopct='%1.2f%%', startangle=140, colors=['red', 'salmon'])  plt.title('Palabras Negativas')  plt.tight\_layout()  plt.show()  Porcentaje de textos con palabras positivas en obama['lematized\_text']: 22.34%  Porcentaje de textos con palabras positivas en df['lematized\_text']: 22.74%  Porcentaje de textos con palabras negativas en obama['lematized\_text']: 19.00%  Porcentaje de textos con palabras negativas en df['lematized\_text']: 15.96% |

* 1. Escoge un tema y clusteriza los usuarios según polaridades.

1. ¿Hay alguna correlación entre la polaridad de un tweet y la fecha en que se publicó?
   1. ¿Los tweets publicados durante ciertos períodos de tiempo tienden a ser más positivos o negativos que otros?
2. Identifica los Top 10 Trolls y Top 10 Influencers. Justifica las características de un usuario Troll e Influencer.
3. **Extra:** Utiliza Transformers con el pipeline de Huggingface para calcular la polaridad de los tweets y comparar los resultados de la pregunta 1.

**VISUALIZACIÓN**:

1. ¿Cómo se distribuyen los tweets según su polaridad a lo largo del tiempo?
2. Visualiza el análisis sintáctico (número de palabras, frase, verbos, nombres…) de los top 10 Trolls e Influencers.
3. ¿Existe alguna correlación entre el número de seguidores de un usuario y la polaridad de sus tweets? Representa visualmente esta relación.
4. Crea una nube de palabras para cada polaridad.
5. ¿Cómo se distribuyen los tweets según su polaridad en función de la hora del día o el día de la semana?