Sentimientos en 280 carácteres

Alicia Sánchez Hossdorf
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
alisanhos@alum.us.es

Rafael Segura Gómez
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
rafseggom@alum.us.es

***Resumen*— El objetivo principal de este trabajo es desarrollar un modelo mediante aprendizaje automático, que sea capaz de analizar los sentimientos de un texto y asi aplicarlo para predecir el estado de ánimo de personajes influyentes a través de sus tweets. Para ello utilizaremos herramienta de procesamiento de lenguaje natural como NLTK [5].**

Con este proyecto hemos podido poner en práctica lo aprendido en clase, pudiendo cumplir todos los objetivos de este proyecto y conseguir resultado satisfactorios.

Palabras Clave—Inteligencia Artificial, aprendizaje automático, procesado de lenguaje natural, análisis de sentimientos …

I. INTRODUCCIÓN

Cada vez hay más necesidad de extraer información a partir de grandes volúmenes de datos de internet. Debido a esto, el análisis de sentimientos se ha vuelto una parte fundamental de la Inteligencia Artificial y del Procesamiento de Lenguaje Natural. El objetivo es analizar las opiniones y emociones expresada por los usuarios en redes sociales, artículos de opinión, etc.

El análisis de sentimiento se aplica en un amplio número de campos. Desde el marketing, hasta la política y salud mental. Por ejemplo, en el ámbito empresarial se usa para conocer la satisfacción de los clientes; en política se usa para ver que opinan las personas ante el discurso del algún candidato; para la salud mental se usa para la detección temprana de depresión, ansiedad u otros trastornos emocionales.

En este proyecto vamos a desarrollar un modelo en Python que analice los sentimientos de un conjunto de Tweets y sea capaz de predecirlos correctamente. Los Tweets son textos cortos de máximo 280 caracteres que se comparten en Twitter, una red social, y con el análisis de datos podremos tener una visión de las opiniones y emociones de los usuarios, etiquetando cada Tweet con: “Muy Feliz”, “Contento”, “Neutro”, “Molesto” y “Hater”.

Para ello vamos a recopilar un conjunto de Tweets, procurando que haya una proporción equilibrada de sentimientos. Deberemos usar herramientas ya existentes para el procesamiento de lenguaje natural, como NLTK [5]; para el análisis de sentimientos, como TextBlob [2]; y finalmente entrenar un modelo que pueda predecir los sentimientos de dos personajes públicos, uno conocido por ser “hater” y otro por tener una valoración social positiva.

II. PRELIMINARES

En esta sección se hace una breve introducción de las técnicas empleadas y también trabajos relacionados, si los hay.

A. Métodos empleados

En este trabajo se usarán diversos métodos y técnicas para desarrollar un modelo capaz de analizar los sentimientos de un conjunto de Tweets. A continuación, describiremos brevemente lo utilizado en el proyecto.

En un comienzo haremos una recopilación de datos, para ellos obtendremos un conjunto de entrenamiento con mas de 1000 Tweets que cumpla una proporción equilibrada sentimientos. Para ellos descargaremos una base de datos de Kaggle [1].

Tras ello deberemos hacer una función para procesar el texto de los Tweets. Para procesar el texto tenemos que eliminar las StopWords, es decir, palabras comunes, eliminar hashtags, URL, menciones y símbolos extraños. Y finalmente usar técnicas de tokenización y stemming. Usaremos la librería NLTK [5] junto al uso de regex para eliminar todo lo que nos piden.

El tercer paso será etiquetar los datos obtenidos tras procesar el texto. Para dicho etiquetado haremos uso del paquete de librería TextBlob [2]. TextBlob nos dará un valor entre el -1 y 1, por tanto, asignaremos a cada etiqueta un rango. Obtendremos un archivo CSV con dos columnas: una con el texto y otra con la etiqueta asignada.

Antes de entrenar el modelo, validaremos la predicción realizada para asegurarnos que haya etiquetado correctamente los conjuntos de entrenamiento y de prueba; Para el entrenamiento de modelo deberemos codificar los atributos y la variable objetivo. Como solo tenemos un atributo y es de tipo texto, usaremos ‘CountVectorizer ()’ [3]. Para la variable objetivo usaremos ‘LabelEncoder ()’ [4].

Entrenaremos tres tipos diferentes de modelo con el conjunto de prueba, al cual habremos aplicado anteriormente td-idf, usando Naive Bayes, Arboles de decisión y Knn.

Despues aplicaremos dichos modelos al conjunto de prueba para comprobar que todo haya funcionado correctamente haciendo uso de la Tasa de acierto.

Finalmente, escogeremos el modelo con mayor tasa de acierto, y analizaremos los Tweets de personajes públicos con fama de Hater y otros con valoración social positiva.

B. Trabajo Relacionado

Antes de comenzar con el proyecto, investigamos un poco por internet encontrando algunas fuentes que nos servirían de ayuda para poder entrenar el modelo:

- Working with Data Text [6]
- NTLK: StopWords [7]

III. METODOLOGÍA

En esta sección entraremos en profundidad en lo descrito anteriormente en preliminares, dividiendo paso por paso.

*1) **Recopilación de datos .***

Debido a las nuevas restricciones que hay respecto al uso de las API de Twitter, no pudimos tener acceso a ella. Por tanto accedimos a Kaggle que dispone de un amplio abanico de DataBase en formato CSV con grandes cantidades de tweets.

Teniamos como objetivo tener un conjunto de entrenamiento con mas de 1.000 tweets, por ello decidimos descargar uno con 27.000. Haciendo uso de la librería Pandas [8]. El archivo CSV contenia mas columnas de las necesarias, así que hicimos una selección se la columna [‘text’] que contenia el tweet.

	text
0	I`d have responded, if I were going
1	Sooo SAD I will miss you here in San Diego!!!
2	my boss is bullying me...
3	what interview! leave me alone
4	Sons of ****, why couldn`t they put them on t...
5	http://www.dothebouncy.com/smf - some shameles...
6	2am feedings for the baby are fun when he is a...
7	Soooo high
8	Both of you
9	Journey!? Wow... u just became cooler. hehe....
10	as much as i love to be hopeful, i reckon the...

Fig. 1. DataBase con tweets originales

*2) **Eliminar Palabras que no aporten información**.*

Para el procesamiendo de los datos tuvimos que descargarnos e importar la librería de NLTK [5], y regex.

Tras crear una lista con las StopWords, es decir, las palabras más usadas y con menos importancia, creamos dos funciones necesarias para la función principal que se encarga de procesar el texto.

- **Delete_Stopwords():** esta función tokeniza el texto del tweet, es decir, separa cada palabra y cada carácter, transformándolo en una lista. Para posteriormente recorrer el tweet tokenizado y comprobar que no sea una StopWord. Si lo es, no lo guarda en la nueva lista la cual devolveremos, con el tweet limpio.

Delete_StopWords()

Entrada:
- Un String tweet

Salidas:
- Una lista de Strings con sin stopwords. L

Algoritmo:

Poner todo el String tweet en minúscula
Tokenizar el String tweet
Crear una lista vacía para el Tweet limpio L
Mientras i < longitud(tweet)
    Si tweet[i] no es stropword
        Añadir tweet[i] a la lista de L
Devolver L.

- **Delete_otherElements():** esta función elimina los hashtags, URL, menciones y símbolos raros mediante el uso de regex.

Delete_OtherElements()

Entrada:
- Un String T

Salidas:
- Un String TL limpio.

Algoritmo:

Eliminar las menciones @ de T
Eliminar los hashtags # de T
Eliminar los URL de T.
Eliminar los símbolos raros de T.
Devolver TL.

- **Clean_Text():** Esta función se encarga de limpiar el todos los tweets haciendo uso de las dos funciones anteriores y lematizar las palabras, es decir, reducirlas a su forma base. Mediante un bucle recorre todos los Tweets del CSV y aplica las funciones, añadiéndolas a una nueva dataBase con tweets completamente limpio con solo las palabras importantes

Clean_Text()

Entrada:
- Un String tweet

Salidas:
- Una lista de Strings con sin stopwords. L

Algoritmo:

Poner todo el String tweet en minúscula
Tokenizar el String tweet
Crear una lista vacía para el Tweet limpio L
Mientras i < longitud(tweet)
    Si tweet[i] no es stropword
        Añadir tweet[i] a la lista de L
Devolver L.

Como resultado obtendremos una database con una lista de palabras correspondiente a los tweets limpios.

	text
0	[respond, go]
1	[sooo, sad, miss, san, diego]
2	[boss, bulli]
3	[interview, leav, alon]
4	[son, put, releas, alreadi, bought]
5	[shameless, plug, best, ranger, forum, earth]
6	[2am, feed, babi, fun, smile, coo]
7	[soooo, high]
8	[]
9	[journey, wow, u, becam, cooler, hehe, possibl]
10	[much, love, hope, reckon, chanc, minim, p, ne...

Fig. 2. Database con Tweets limpios tokenizados.

3) *Etiquetado de datos*.

Para el etiquetado de sentimientos vamos a hacer uso de la librería TextBlob [2]. La función sentiment.polarity() de la librería TextBlob [2] analiza la polaridad de las palabras, clasificándola con valor numérico entre -1 y 1. Con ésto conseguimos parte del objetivo, ver si las palabras usadas son positivas o negativas y como de polarizadas estan.

- **tag_text():** Con esta función aplicamos la función sentiment de la librería textblob [2], concretamente el cálculo de la polaridad para darle un valor entre -1 y 1.

tag_text()

Entrada:
- Lista de String

Salidas:
- Tupla [texto, etiqueta]

Algoritmo:
1. Convierte la lista de tokens en texto
2. Convierte la lista en el tipo blob
3. Aplicamos sentiment.polarity
4. Devolvemos la tupla

Hemos concatenado esta función con la escritura en archivo, concretamente en formato csv. Comentaré la escritura por encima ya que el código está comentado y es una escritura simple, nada especial, usada varias veces de forma parecida durante todo el proyecto.

En este caso creamos un csv nuevo al que llamaremos *resultados.csv*, con formato 'UTF-8-sig' porque, a base de prueba y error, es el que menos problemas presenta. Escribimos las cabeceras, '*text*,'*tag*' y '*sentiment*', comprobamos que efectivamente la variable '*tweets_cleaned*' tenga algo y leemos fila por fila de la variable usando *iterrows*, ya que es de tipo Dataframe, y las vamos guardando en el archivo csv ya creado.

- **etiqueta_codificada():** Función creada para clasificar la polaridad en los valores discretos que se nos presenta en el enunciado: *[hater, molesto, neutro, contento, muy feliz]*. Hemos decidido, un poco arbitrariamente, que al ser -1 el más enfadado le damos la categoría de hater y vamos en intervalos de 0.5, siendo 0 el neutro.

etiqueta_codificada()

Entrada:
- Etiqueta (valor numérico/ordinal)

Salidas:
- Sentimiento (valor discreto)

Algoritmo:
5. Lee la etiqueta, su valor numérico
6. Mediante if-else comprueba valores:
   a. [-1,-0.5) → hater
   b. [-0.5,0) → molesto
   c. (0,0) → neutro
   d. (0,0.5] → contento
   e. (0.5,1] → muy feliz
7. Devolvemos la "etiqueta" discreta correspondiente.

4) *Entrenamiento del Modelo y predicción de Tweets.*

Antes de comenzar con el entrenamiento, debemos preparar los datos. Para ello deberemos codificar tanto el texto (atributo) como los sentimientos (el objetivo).

Normalmente para codificar los atributos usaríamos ordinalEncoder(). Pero nosotros lo que tenemos son atributos de tipo texto, no discreta, a pesar de ser ambos string. Por tanto, usaremos CountVectorizer(), el cual transformara nuestro la columna ['text'] en un vector disperso. Dicho vector tiene forma (*x*, *y*) *z*, donde *x* es el índice del tweet, *y* es el índice de la palabra y *z* es cuantas veces aparece *y* palabra en *x* tweet

```
(0, 16741)    1
(0, 8957)     1
(1, 18398)    1
(1, 17157)    1
(1, 13294)    1
(1, 17233)    1
(1, 6493)     1
(2, 3969)     1
(2, 4296)     1
(3, 10745)    1
(3, 11925)    1
(3, 2305)     1
```

Fig. 3. Vector del texto.

Pera ello importaremos la librería feature_extraction.text de sklearn.

Para codificar el objetivo usaremos labelEncoder(), importando la librería preprocessing de sklearn. Codificamos

la columna [‘sentiment’] ya que es el objetivo y asi obtenemos una lista con números que representan la etiqueta del sentimientos, con valores del [0, 4].

```
[4 2 4 ... 3 0 0]
```

Fig. 4. Lista objetivo codificado.

Tras haber codificado el conjunto, debemos separarlo para obtener un conjunto de entrenamiento y otro de prueba. Para ello se hace uso de train_test_split(), separando los datos en un 66% para entrenamiento y un 33% para prueba.

Por último, debemos aplicar td-idf ya que qué habrá tweets con muchas mas palabras que otras, dándole mas relevancia a aquellas palabras que sean mas recurrentes, pero disminuyendo a las menos comunes. Para ello, usamos TfidfTransformer() sobre el conjunto de atributos de entrenamiento.

a) Naives Bayes.

Para el entrenamiento del modelo usando Naive Bayes, no trabajaremos con CategoricalNB() ya que tenemos un volumen muy grande de datos y para ello deberíamos crear una matriz densa. Por ello, usaremos MultinomialNB() [9] que nos permite usarlo con el vector disperso que ya tenemos, además de que funciona mucho mejor con conjunto de datos grandes, usando suavizado.

Para ello entrenamos el modelo usando la función fit(), metiéndole como parámetros los atributos de entrenamiento y el objetivo de entrenamiento.

Tras ello, comprobaremos con los datos de prueba si predice correctamente los sentimientos, pudiendo calcularse la tasa de acierto con dos funciones diferentes: modelNB.score() o accuracy_score()

```
Tasa de acierto:  0.7706472598963502
```

Fig. 5. Tasa de acierto NB

b) Knn.

Para Knn usaremos KNeighborsClassifier() [10]. Usaremos el numero de vecinos que viene por defecto, que es n = 5, y la métrica de coseno ya que como indica la api, es la mejor para clasificar texto.

Como hicimos anteriormente, entrenamos el modelo con la función fit() y comprobamos la tasa de acierto al aplicar el modelo con los datos de prueba:

```
Tasa de acierto:  0.6572940787297387
```

Fig. 6. Tasa de acierto Knn

c) Arbol de decisión.

Finalmente, para árbol de decisión usaremos DecisionTreeClassifier() [11]. Para este modelo hemos usado comparación cruzada, ya que podiamos elegir cual era la maxima profuncidad del modelo y que tipo de criterio seguir para que tuviese mejor tasa de acierto. Probamos con profundidad maxima 3, 5, y 10. Y con el criterio Giny o criterio Entropy.

En el codigo solo hemos dejado dos pruebas. Una con profundidad 3 y criterio Entropy y otra con profundidad 5 y criterio 5.

Como hicimos anteriormente, entrenamos el modelo con la función fit() y comprobamos la tasa de acierto al aplicar el modelo con los datos de prueba:

```
Tasa de acierto:  0.5693020178630499
```

Fig. 7. Tasa de acierto Árbol de decisión – prueba 1

```
Tasa de acierto:  0.608556621457713
```

Fig. 8. Tasa de acierto Árbol de decisión – prueba 2

5) ***Analisis de Tweets de personajes públicos***.

Hemos visto, analizando cada modelo y su tasa de acierto independientemente, pero con la misma semilla, para garantizar la máxima igualdad posible al entrenar los modelos, que el modelo con mayor tasa de acierto es Naive Bayes, con un acierto del 77’06%.

Procedemos al análisis de tweets. En este apartado se recomendó usar Twint para descargar los tweets de los usuarios objetivo de análisis, pero los recientes cambios en la API de Twitter y en la estructura de la propia web han dejado la herramienta inservible, así que hemos optado por crear un script, a mano y desde cero, usando herramientas de scrapping como *Selenium* [12] y *BeautifulSoup4* [13] para conseguir nuestro propósito.

- **scrape_tweets():** Esta función crea un array vacío donde guardaremos los tweets, abre la web de Twitter en el perfil del usuario que hayamos introducido como variable y, mediante automatización con *selenium* espera a que la web cargue (aproximadamente 3 segundos de ‘sleep’), hace scroll hasta el final de la página y vuelve a esperar a que cargue, guardando los tweets que encuentre en el proceso haciendo uso de *beautifulsoup4*. Al final de esta celda del notebook está el uso del script, escribiendo los ‘*usernames*’ que deseemos y el número de tweets a ‘*scrappear*’.

  También se incluye una escritura en archivo de formato *.csv*, guardando éste en la carpeta

'influencers', con un formato similar al archivo de los datos de entrenamiento.

scrape_tweets()

Entrada:

- nombre de usuario, cantidad

Salidas:

- archivo csv con los tweets

Algoritmo:

8. Crea el array vacío donde guardaremos los tweets
9. Carga el perfil del usuario deseado y espera 3 segundos
10. Hace scroll hasta el final de la página
11. Atendiendo al formato de la web:
    a. Encuentra el '*div*' con el tweet
    b. Parsea con beautifulsoup4
    c. Guarda en el array inicial
12. Cuando ya tiene 100 tweets, termina automatización
13. Crea directorio destino
14. Crea el archivo csv, con header, y lo guarda en destino

Tras conseguir los Tweets, se aplica el modelo elegido y ya entrenado, en este caso Naive Bayes, pero no podemos aplicarlo tal cual, ya que los tweets no están "depurados" como el conjunto de entrenamiento, necesitamos aplicarle el mismo procedimiento. Tras 'limpiar' los tweets los convertimos, de vuelta, en una cadena de texto.

Entrenamos el vectorizador con los tweets originales, los datos de entrenamiento. Con ese vectorizador ya podemos codificar los textos y aplicarle el modelo correctamente, representando los valores discretos (la inversa de la transformación del objetivo) en un *pie chart.*

IV. Resultados

En nuestro proyecto, al haber conseguido automatizar esta tarea, hemos hecho análisis de 11 personas, siendo bastante trivial eliminar o añadir nuevos usuarios a analizar, lo único que dificulta el proceso es esperar a que la automatización del scrapping consiga sacar todos los tweets.

Durante la realización del proyecto hemos probado con diferentes database de diferentes tamaños. La mas pequeña tenia 79 tweets y comprobamos que el mejor modelo era Arboles de Decisión, usando el criterio de Entropia y profundidad 10, pero la tasa de acierto era muy baja por la poca cantidad de datos ≈ 43%.

Después de cambiar nuestro database al original, el mejor modelo cambió a ser Naive Bayes, con una tasa de acierto más alta: 77%.

Con el modelo ya entrenado pasamos a realizar el experimento de analizar los sentimientos de personas influyentes en internet.

Hemos elegido a @pontifex, el Papa Francisco, como persona con buena reputación, el cual no debería tener contenido tóxico.

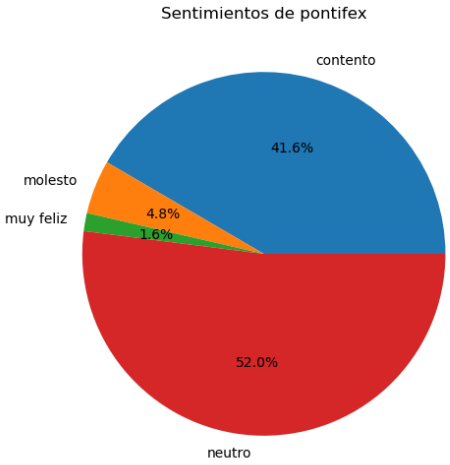

Fig. 9. Pie Chart del Papa Francisco

Como persona con mala reputación hemos escogido a Elon Musk, @elonmusk, el dueño de Tesla y actualmente dueño de Twitter. Es conocido por sus discursos de odio y las faltas de respeto hacia otros usuarios.

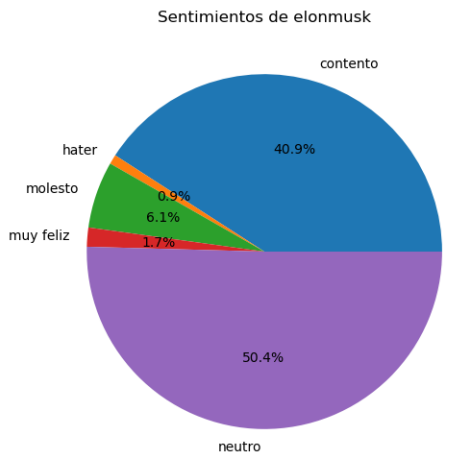

Fig. 10. Pie Chart de Elon Musk

Como experimento social hemos querido analizar la influencia de ciertas personas, como streamers, en los jóvenes. Para ello hemos elegido a @xQc, uno de los más conocidos en EEUU.

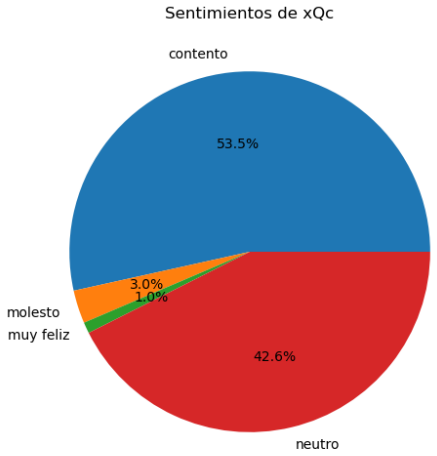

Fig. 11. Pie Chart de xQc.

Y para concluir, analizamos el contenido de una cuenta de noticias mundiales, tales como ataques, guerras, política, etc.

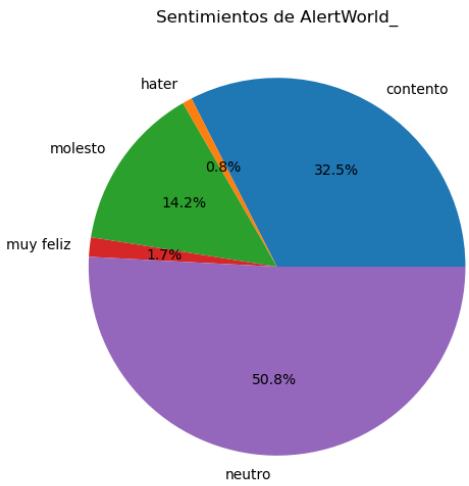

Fig. 12. Pie Chart de Alert World.

Comparamos al Papa con Elon Musk como personas de reputaciones opuestas, el Papa positiva y Elon Musk negativa, pero sus análisis nos indican resultados un poco confusos. Ambos poseen un alto porcentaje de tweets neutros, los cuales ignoramos, pero vemos que el Papa sí tiene tweets donde se encuentra muy feliz, ausentes en Elon Musk, aunque Elon tiene mayor porcentaje de tweets contento, pero mayor porcentaje de molesto y la existencia de tweets haters, aunque sea nimia (1%), pero el Papa no tiene ninguno como hater. Por lo general diríamos que el Papa es más neutro en sus tweets con algunos de ellos muy positivos mientras que Elon, por lo general, parece más positivo pero con un pequeño porcentaje de tweets muy negativos.

Al analizar al influencer de internet, Felix ‘xQc’ Lengyel, observamos que la mayoría de sus tweets se clasifican como contentos, con un porcentaje muy pequeño de molesto. Tampoco se le conoce por ser una persona conflictiva por lo que los resultados son lo esperado, buena reputación.

Y, por último, analizar el perfil de noticias AlertWorld_, dedicado a subir avisos y tweets de eventos recientes en el mundo, muchos de ellos por desgracia siendo sucesos bélicos. Esto se refleja en el análisis, podemos observar que hay un 10.9% de molesto, un porcentaje alto en comparación con otros análisis y con otros sentimientos. Que el porcentaje de neutro sea tan alto también indica la neutralidad de la información, buscando no polarizar ni influir en los sentimientos.

V. Conclusiones

Se ve de un simple vistazo que la mayoría de los tweets usan palabras clasificadas como neutras, que no expresan opinión o sentimientos, en la mayoría de análisis el porcentaje de neutro supera el 40%. Observamos que perfiles como el de Elon Musk, de reputación negativa, no tiene un porcentaje de tweets negativos demasiado alto, como sería de esperar. Eso nos lleva a pensar que se expresa usando palabras neutrales o no de odio, aunque las ideas detrás sean realmente negativas, pero es el problema de analizar por tokens o palabras y no por el sentido real de la frase.

Lo mismo ocurre con perfiles como @cobratate (Andrew Tate, famoso por ser misógino, proxeneta, machista, xenófobo, racista y recientemente encarcelado por tráfico de personas). Su porcentaje de tweets negativos sí es de los más altos de los analizados, pero sigue sin superar el 10%.

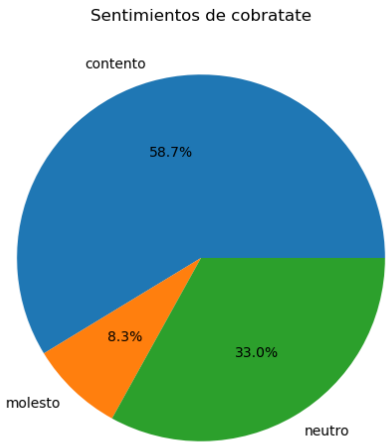

Fig. 13. Pie Chart de Cobratate.

Y es curioso observar que ambos perfiles de noticias como BBC News y AlertWorld tienen un porcentaje superior al 10% en ‘molesto’, lo cual nos
lo cual nos lleva a pensar que las noticias actuales no son demasiado buenas.

VI. Referencias

[1] «Kaggle». URL: https://www.kaggle.com/datasets/yasserh/twitter-tweets-sentiment-dataset.

[2] «TextBlob» URL: https://textblob.readthedocs.io/en/dev/.

[3] «API CountVectorizer()» URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[4] «API LabelEncoder ()» URL:https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[5] «NTLK» URL: https://www.nltk.org/.

[6] «Working With Data Test» URL: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html.

[7] «StopWords» URL: https://pythonspot.com/nltk-stop-words/.

[8] «API Pandas » URL: https://pandas.pydata.org/docs/reference/index.html.

[9] «API MultinomialNB()» URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[10] «API KNeighborsClassifier()» URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[11] «API DecisionTreeClassifier ()» URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[12] «Selenium,» URL: https://www.selenium.dev/documentation/.

[13] «BeautifulSoup» URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.