ANÁLISIS SENTIMENTAL DURANTE LA PANDEMIA POR COVID-19 (SARS-COV-2) BASADO EN LA RED SOCIAL TWITTER

Christian Yesid Galindo Cano
Universidad Distrital Francisco José de Caldas

***Resumen*—El proyecto consiste en realizar una aplicación web donde se pueda ver la evolución de los sentimientos de los usuarios de la red social Twitter durante la pandemia por el virus Covid-19. Se hace uso del lenguaje de programación Python y sus librerías como NumPy, Pandas entre otras para el análisis de datos. Para la extracción, se utiliza la librería twint y para la interpretación de los datos o tuits seleccionados (que deben tener relación con la pandemia) se utiliza la librería TextBlob. Para el despliegue del aplicativo web se utiliza Flask (framework escrito en Python) además de HTML, CSS y JS.**

El usuario final puede realizar búsquedas con diferentes filtros aplicados a los tuits por ejemplo las fechas dentro de las que se van a analizar (en relación con el periodo de tiempo de la pandemia por coronavirus), el lenguaje en el que se encuentran escritos y si son seleccionados de un usuario en específico o de una muestra de usuarios aleatoriamente.

Palabras clave—Análisis, Sentimientos, Python, Twitter.

***Abstract*-- The project consists of creating a web application where you can see the evolution of the feelings of users of the social network Twitter during the pandemic due to the Covid-19 virus. The Python programming language and its libraries such as NumPy, Pandas among others are used for data analysis. For the extraction, the twint library is used and for the interpretation of the selected data or tweets (which must be related to the pandemic) the TextBlob library is used. For the deployment of the web application, Flask (framework written in Python) is used in addition to HTML, CSS and JS.**

The end user can perform searches with different filters applied to the tweets, for example the dates within which they will be analyzed (in relation to the time period of the coronavirus pandemic), the language in which they are written and if They are selected from a specific user or from a random sample of users.

Keywords—Analysis, Feelings, Python, Twitter.

I. Introducción

La pandemia causada por coronavirus ha afectado todos los ámbitos de la sociedad como el de la salud, económico, político, deportivo, de entretenimiento, entre otros. Debido a esto los gobiernos han tenido que decretar medidas tanto para evitar la propagación del virus, como para reactivar la economía. Una de las principales medidas adoptadas por los diferentes gobiernos para evitar los contagios, es el confinamiento, donde la población ha tenido que permanecer en sus lugares de residencia, lo cual ha disminuido la interacción social habitual.

Este aislamiento, el miedo a contagiarse o que alguien cercano se contagie, la situación económica y la imposibilidad de llevar a cabo las actividades cotidianas diferentes al trabajo o estudio, por ejemplo, salir a comer a un restaurante, ir al gimnasio, practicar algún deporte con amigos, puede generar cambios de emociones y a su vez cambios en las interpretaciones de estas, conocidas como sentimientos.

De acuerdo a [1], la mayor preocupación de los expertos es hacer frente a la ansiedad, que es causada por un sentimiento de miedo, temor e inquietud. Se tiene en cuenta la ansiedad generada por la preocupación de contagio y la ansiedad no tratada derivada de otras enfermedades, por ejemplo, personas con esquizofrenia que deben salir a hacer ejercicio físico y por la situación actual no es fácil realizarlo. Por otra parte, los resultados de las encuestas realizadas a la población China al inicio de la epidemia (aún no era considerada pandemia) [2], muestran un alto porcentaje de personas que presentan ansiedad y síntomas de depresión. Además, los investigadores encontraron que las emociones negativas aumentaron como la ansiedad, la depresión y la ira, mientras que las emociones positivas y la satisfacción con la vida disminuyó.

Es por ello que resulta interesante realizar un estudio de los sentimientos que expresan las personas durante la pandemia a través de las redes sociales, esto se logra mediante el procesamiento del lenguaje natural, que es un área de conocimiento de la inteligencia artificial encargado de la comunicación de la máquina con el humano y funciona mediante machine learning. Específicamente se aplican técnicas para buscar patrones o tendencias que brinden información adicional no explícita en grandes cantidades de texto, estas técnicas junto a un proceso de análisis es lo que se conoce como minería de textos.

Esta tecnología sirve para realizar diferentes análisis, por ejemplo: sentimental, los temas de tendencia, o la intención de voto; Por ejemplo, en [3], donde realizan un análisis del apoyo a los partidos políticos durante los períodos electorales de 2015 y 2016 en España a partir de los comentarios de Twitter.

El análisis sentimental basado en redes sociales ya se ha aplicado, por ejemplo, para la clasificación de personalidad proactiva basado en Weibo [4], o el análisis de sentimientos negativos en las redes sociales en China a través del modelo BERT [5] basado también en Weibo.

Teniendo en cuenta que estas aplicaciones para realizar análisis sentimental están enfocadas a la utilización de expertos, se hace necesario la creación de una aplicación web que permita realizar análisis sentimental basado en la pandemia con diferentes filtros que pueda ser utilizado por un usuario común.

II. Objetivos

- *Obtener los tuits usando la librería de Python Twint.*
- *Brindar diferentes porcentajes de estados de ánimo basados en los tuits.*
- *Mostrar los resultados a través de una interfaz gráfica amable con el usuario*

III. Trabajos Relacionados

A. Análisis sentimental basado en partes del discurso

Los métodos tradicionales de análisis de sentimientos basado en texto, se enfocan principalmente en las palabras dejando de lado la información que puede contener la semántica del discurso completo. Por esta razón se presentan mejoras a los modelos existentes o implementación de nuevos modelos que superen la eficiencia de los anteriores.

Por ejemplo en [6], se propone la construcción del léxico de sentimientos, basado en fragmentos de parte del discurso, part-of-speech (POS), dependientes del contexto, a los que llaman CP-chunks. De esta manera se pretende resolver la ambigüedad en el léxico de los sentimientos. Los resultados de este nuevo método indican que, en comparación con los métodos existentes, es más estable y equilibrado para los corpus de polaridad positiva y negativa.

Por otra parte, se hace uso de redes neuronales basadas en la atención para tareas de clasificación de sentimientos [7], en este artículo se propone una red de atención transformadora basada en la parte del discurso, part-of-speech Transformer Attention Network (pos-TAN). Este modelo utiliza el mecanismo de auto-atención para aprender la expresión de características del texto, y también incorpora la atención POS, para capturar la información sentimental contenida en la parte del discurso. Este modelo es aplicado a diferentes conjuntos de datos donde los resultados indican la eficacia de este.

Una aplicación basada en reseñas de clientes [8], presenta una nueva implementación del método de identificación de polaridad de palabras basado en el léxico en varios conjuntos de datos. Esta variación opera calculando la relación semántica entre el conjunto de expansión de contexto de la palabra de destino y un conjunto de expansión de sinónimos que comprende los sinónimos de todas las palabras que rodean la palabra de destino dentro del texto original. La polaridad de la palabra de destino se determina como aquella para la que la relación semántica entre estos dos conjuntos significativos es la más alta. Los resultados de la implementación demuestran que la variación utilizada del método de identificación de la polaridad de las palabras basado en el léxico se comporta de forma favorable frente a los métodos comparados.

B. Minería de textos aplicada a redes sociales

La minería de textos aplicada a redes sociales sirve para realizar estudios en diferentes ámbitos, por ejemplo, para para el análisis de los partidos de fútbol [9], en este artículo se utiliza la minería de textos y el análisis clúster para identificar oportunidades que mejoren el rendimiento del equipo, en base a los comentarios de los aficionados. Se realizó un estudio de caso de la clasificación final de la Copa Mundial de la FIFA 2018 de Corea, Corea versus Uzbekistán. El análisis sobre los comentarios recopilados de los fanáticos en el estudio reveló 16 oportunidades diferentes que satisfarían a los fanáticos con respecto al rendimiento del equipo, y de esas, se identificaron dos oportunidades extremas principales.

En [10] se hace un estudio donde se identifican los temas dominantes de la investigación basada en Twitter. Este estudio recopiló artículos relevantes de tres bases de datos y aplicó la minería de textos y el análisis de tendencias para detectar patrones semánticos y explorar la evolución anual de los temas de investigación a lo largo de una década. En los resultados se encuentran 38 temas en más de 18.000 manuscritos publicados entre 2006 y 2019. Al cuantificar las tendencias temporales, este estudio encontró que mientras que el 23,7% de los temas no mostró una tendencia significativa (P = 0,05), el 21% de los temas tuvo tendencias crecientes y el 55,3% de los temas tuvo tendencias decrecientes.

Otro ejemplo de aplicación en Twitter, es el estudio para medir el apoyo político durante los periodos electorales a partir de los comentarios [3], donde se incluyen 250.000 tuits sobre las elecciones generales españolas de 2015 y 2016, respectivamente. La información es obtenida de tres regiones españolas definidas por geolocalización, además se seleccionan rasgos basados en palabras clave de los cuatro principales partidos políticos. Los resultados de este análisis indican un comportamiento de apoyo constante de los usuarios hacia los partidos tradicionales y un comportamiento optimista de los usuarios con respecto a los partidos políticos emergentes. Para el análisis de sentimientos durante procesos electorales también se ha usado minería de patrones [11], para descubrir patrones y pautas de sentimiento en textos de microblogging, se emplea un conjunto inicial de 1,7 millones de tuits para analizar los sentimientos más destacados durante la campaña preelectoral estadounidense y los resultados obtenidos avalan la capacidad del sistema de obtener reglas de asociación y patrones con gran valor descriptivo en este caso de uso.

Otra red social donde se realizan estudios de análisis sentimental es Weibo (red social china similar a Facebook), por ejemplo, para la clasificación de personalidad proactiva [4], se seleccionan 901 participantes y se analiza sus publicaciones mediante cinco algoritmos de aprendizaje automático y siete indicadores. Otro ejemplo de estudio es la extracción de las causas de la ideación suicida (SICE) en los textos sociales [12] que pueden servir de apoyo para la prevención del suicidio. Esta investigación se hace mediante el uso de análisis psicológicos y sociológicos. Por último, el proyecto más similar al que se va a realizar es el análisis de sentimientos negativos en las redes sociales en China a través del modelo BERT enfocado al Covid-19 [5], donde se analizan 999.978 publicaciones relacionadas con COVID-19

seleccionadas al azar entre el 1 de enero y el 18 de febrero de 2020. Para clasificar las categorías de sentimiento (positivo, neutro y negativo) se utiliza el modelo BERT y se utiliza el modelo TF-IDF para resumir los temas de las publicaciones. Los resultados indican que las principales preocupaciones de la población con respecto al Covid-19 son: el origen del virus, síntomas, actividades de producción y el control de salud pública.

IV. Metodología

Para el desarrollo de este proyecto se aplica modelo en cascada, el cual se compone de las fases que se describen a continuación:

1. Análisis: Se realiza la planeación y la especificación de requerimientos
2. Diseño: Especificación del sistema (arquitectura de software)
3. Implementación: Programación del sistema y pruebas
4. Verificación: Integración del sistema
5. Mantenimiento: Entrega del sistema y mejora.

Cada una de estas fases se ejecuta una sola vez y cada una depende de la anterior (a excepción de la fase de análisis) como se muestra en la figura 1.

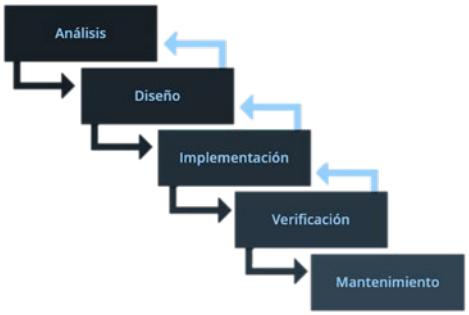

Fig 1. Modelo Cascada [13]

V. Análisis

A. Descripción detallada del problema

Se busca realizar un sistema de análisis de sentimiento basado en Twitter desarrollado con el lenguaje de programación Python. Para ello se debe tratar con grandes cantidades de información y hacer operaciones sobre esta. Los requisitos del sistema son los siguientes:

1) Búsqueda por filtros

a) El usuario puede realizar la búsqueda con las palabras que desee

b) El usuario podrá realizar búsquedas de usuarios al azar o de usuarios en específico.

c) Como el enfoque del análisis es durante la pandemia, el usuario puede seleccionar las fechas en las que desea realizar la búsqueda, teniendo un rango desde inicios de 2019 (normalidad) hasta la actualidad.

d) El usuario puede seleccionar el lenguaje en el que están escritos los tuits.

e) También puede seleccionar la región en la que desea realizar la búsqueda

2) Despliegue de información

a) El sistema muestra diferentes gráficas que soportan los resultados obtenidos de la búsqueda.

B. Estructura de la aplicación

1) Interfaz de usuario

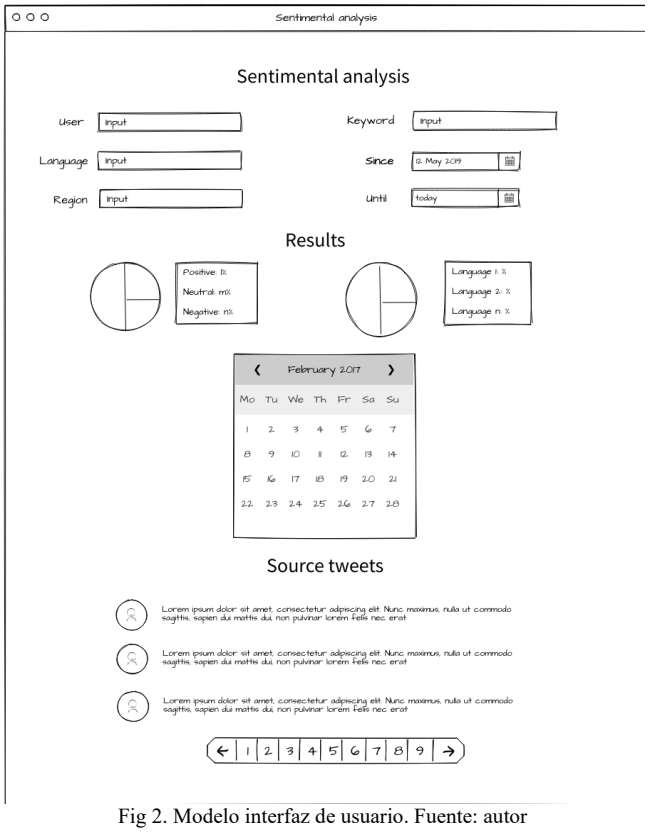

Fig 2. Modelo interfaz de usuario. Fuente: autor

2) Flask

Este apartado es el que se encargará de manejar toda la lógica de la aplicación, tiene el controlador, el modelo y los diferentes componentes para la interfaz (vista).

VI. Diseño

En este apartado se detalla el flujo de la aplicación:

A. Acceso

Para el control de acceso no se utilizará autenticación puesto que cualquier usuario puede realizar consultas en la aplicación.

B. *Realización de una búsqueda*

Para realizar una búsqueda se descargarán n tuits los cuales se pueden filtrar por usuarios, fecha, y palabras clave. La estructura de los tuits obtenidos se detallará en el apartado de desarrollo.

Una vez se obtienen los tuits se pasa a la etapa de análisis. En esta etapa se asigna una polaridad a cada uno de estos siendo Positiva (Polaridad > 0), Neutral (Polaridad igual a 0), y Negativa (Polaridad < 0). Una vez completado este proceso se pasa a la etapa de resultados.

C. *Resultados*

En esta fase se muestran los siguientes diagramas:

- Torta de polaridades: Diagrama que muestra los porcentajes de cada una de las polaridades.
- Torta de lenguajes: Diagrama que muestra los porcentajes de cada uno de los lenguajes encontrados en los tuits.
- Mapa de regiones: Diagrama opcional, es decir, solamente aparece si se especifican regiones.
- Calendario (de sentimientos): Diagrama que muestra un calendario con un respectivo color dependiendo de la polaridad en el día seleccionado.

VII. Desarrollo

Para iniciar con el desarrollo del proyecto se utiliza Python 3.9.7 y se crea un entorno virtual (python -m venv twintv2-env), luego se instalan las siguientes librerías (pip install nombre_librería):

- Flask
- Flask-Cors
- TextBlob
- Twint

A. *Flask*

Flask es un marco de aplicación web WSGI (Web Server Gateway Interface) ligero. Está diseñado para que la puesta en marcha sea rápida y sencilla, con la capacidad de escalar a aplicaciones complejas. Comenzó como una simple envoltura alrededor de Werkzeug y Jinja y se ha convertido en uno de los marcos de aplicaciones web de Python más populares [14].

Flask ofrece sugerencias, pero no impone ninguna dependencia o diseño del proyecto. Depende del desarrollador elegir las herramientas y bibliotecas que desea utilizar. Hay muchas extensiones proporcionadas por la comunidad que facilitan la adición de nuevas funciones [14].

La estructura básica para iniciar Flask se presenta a continuación:

```
2   from flask import Flask, render_template, request
3   from flask_cors import CORS
4
5   app = Flask(__name__)
6   CORS(app)
7
8   @app.route('/')
9   def index():
10      return render_template('index.html')
```

Fig 3. Estructura básica flask. Fuente: Autor

Como se puede observar se realiza la importación de Flask, además de los siguientes componentes:

- request: El contenido que un cliente web manda al servidor siempre va almacenado en la Request. En Flask la Request se representa mediante el objeto request. Para poder utilizar el objeto request se debe importar al principio del programa Flask [15].
  Para el caso de este proyecto se utilizan parámetros de tipo GET, que vienen como una lista de clave/valor en la url de petición y para acceder a estos se hace como se muestra a continuación:

```
18  #/get?query=valor&lang=valor2&place=valor
19  @app.route('/get', methods=['GET', 'POST'])
20  def get():
21      query = request.args.get('query')
22      lang = request.args.get('lang', 'any')
23      place = request.args.get('place', 'any')
24      fetched_tweets = get_tweets(query, lang, place)
25      pol = get_polarity(fetched_tweets)
26      return pol
```

Fig 4. Ejemplo request. Fuente: autor

- render_template: Se utiliza para renderizar un HTML, ya que generar HTML desde Python no es sencillo. Solo se debe indicar el nombre de la plantilla y la lista de variables que se desean pasar al motor de plantillas como argumentos de palabras clave.
-

```
146 @app.route('/')
147 def index():
148     keyword_form = forms.KeywordForm()
149     return render_template('index.html', form=keyword_form)
```

Fig 5. Ejemplo render_template. Fuente: autor

- flask_cors: Este paquete expone una extensión Flask que por defecto habilita el soporte CORS en todas las rutas, para todos los orígenes y métodos. Permite la parametrización de todos los encabezados CORS en un nivel por recurso. El paquete también contiene un decorador, para aquellos que prefieren este enfoque [16].

B. *TextBlob*

Es una biblioteca de Python (2 y 3) para procesar datos textuales. Proporciona una API simple para sumergirse en tareas comunes de procesamiento del lenguaje natural (NLP), como el etiquetado de parte del discurso, la extracción de frases nominales, el análisis de sentimientos, la clasificación, la traducción y más [17].

Con textblob se obtiene una métrica de polaridad y subjetividad. La polaridad es el sentimiento mismo, que va de -1 a +1. La subjetividad es una medida del sentimiento siendo objetivo a subjetivo, y va de 0 a 1 [18].

En el proyecto se utiliza la parte de polaridad y se usa así:

```python
from textblob import TextBlob
def sentiment(text):
    analysis = TextBlob(text)
    if analysis.sentiment.polarity > 0:
        return "Positive"

    if analysis.sentiment.polarity == 0:
        return "Neutral"

    if analysis.sentiment.polarity < 0:
        return "Negative"
```

Fig 6. Uso de textblob en el proyecto. Fuente: autor

A cada tuit obtenido se le aplica la función 'sentiment' para obtener su polaridad.

C. Twint

Twint es una herramienta avanzada de raspado(scraping) de Twitter escrita en Python que permite extraer tuits de perfiles de Twitter sin usar la API de Twitter.

Twint utiliza los operadores de búsqueda de Twitter para permitirle raspar Tuits de usuarios específicos, raspar tuits relacionados con ciertos temas, hashtags y tendencias, o clasificar información confidencial de tuits como correos electrónicos y números de teléfono.

Twint también realiza consultas especiales a Twitter, lo que le permite también rastrear los seguidores de un usuario de Twitter, los tuits que le han gustado a un usuario y a quién siguen sin ninguna autenticación, API, Selenium o emulación de navegador [19].

Para instalarlo se hace con el comando pip install twint, a continuación, se presenta un ejemplo de uso:

```python
import twint

# Configure
c = twint.Config()
c.Username = "realDonaldTrump"
c.Search = "great"

# Run
twint.run.Search(c)
```

Fig 7. Ejemplo de uso twint [19]

En este ejemplo se crea un objeto de tipo twint (c=twint.Config()), luego se define el usuario al que se le va a realizar la búsqueda (c.Username = "realDonaldTrump"), después se define la palabra clave a tener en cuenta cuando se filtren los tuits (c.Search= "great"). Por último, se ejecuta la búsqueda con el comando twint.run.Search(c).

D. Estructura de la aplicación

Luego de ver las bases y cómo funcionan sus principales módulos, se va a mostrar la estructura de la aplicación y cómo trabajan en conjunto estos módulos.

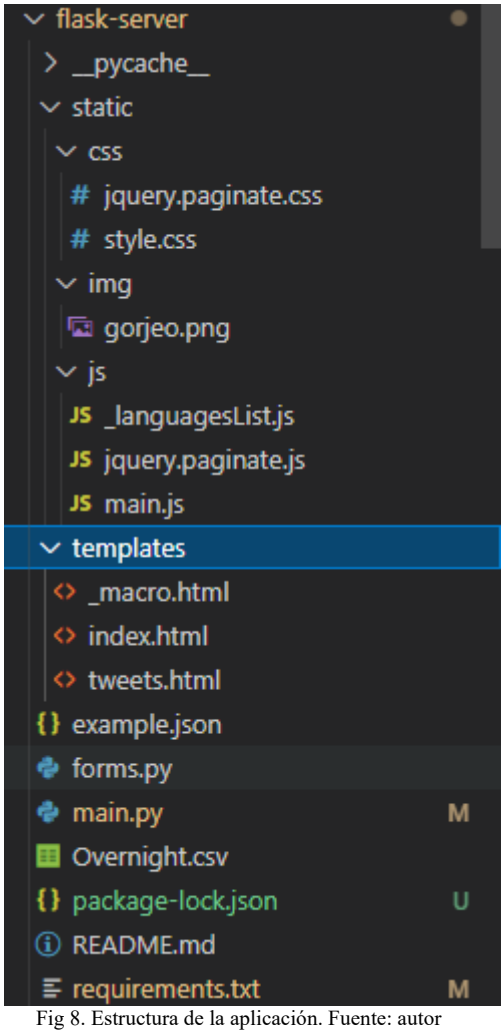

Fig 8. Estructura de la aplicación. Fuente: autor

Como se puede observar se tienen las carpetas static y templates, también los archivos forms.py, main.py example.json, Overnight.csv, package-lock.json, README.md y requeriments.txt, a nivel raíz. Se explicará cada uno de estos en orden de prioridad.

En primer lugar, se tiene el archivo requeriments.txt este archivo contiene las librerías necesarias para poder ejecutar el proyecto:

```
Flask
textblob
Flask-Cors
```

Fig 9. requeriments.txt. Fuente: autor

Luego está el README.md del repositorio que se adjuntará al final de este artículo para que cualquier usuario pueda consultarlo. También el package-lock.json que solamente contiene el número de la versión de la aplicación en este caso 1.

Por otra parte, los archivos example.json y Overnight.csv fueron generados con una búsqueda de ejemplo para verificar que se pueden guardar los tuits extraídos en archivos con extensión json y csv.

```
{"id": 1434202728977158144, "conversation_id": "1434047236007100419",
{"id": 1434202725395451909, "conversation_id": "1434202725395451909",
{"id": 1434202723851902978, "conversation_id": "1433929768974499840",
{"id": 1434202722891337738, "conversation_id": "1434194250791587846",
{"id": 1434202721490509827, "conversation_id": "1434202721490509827",
{"id": 1434202720093843459, "conversation_id": "1434199579491176452",
```


Fig 10. Muestra del archivo example.json. Fuente: autor


```
id,conversation_id,created_at,date,time,timezone,user_id,username,name,
1435392436424085505,1435392436424085505,"2021-09-07 18:59:59 Hora est.
1435754824784482306,1435420019031109633,"2021-09-08 18:59:59 Hora est.
1436117211228237839,1436117211228237839,"2021-09-09 18:59:59 Hora est.
1436479599945428992,1436371225744494607,"2021-09-10 18:59:59 Hora est.
1436841988700217346,1436831473252945920,"2021-09-11 18:59:59 Hora est.
1435392436424085505,1435392436424085505,"2021-09-07 18:59:59 Hora est.
1435754824784482306,1435420019031109633,"2021-09-08 18:59:59 Hora est.
1436117211228237839,1436117211228237839,"2021-09-09 18:59:59 Hora est.
1436479599945428992,1436371225744494607,"2021-09-10 18:59:59 Hora est.
1436841988700217346,1436831473252945920,"2021-09-11 18:59:59 Hora est.
1435392436424085505,1435392436424085505,"2021-09-07 18:59:59 Hora est.
1435754824784482306,1435420019031109633,"2021-09-08 18:59:59 Hora est.
1436117211228237839,1436117211228237839,"2021-09-09 18:59:59 Hora est.
1436479599945428992,1436371225744494607,"2021-09-10 18:59:59 Hora est.
1436841988700217346,1436831473252945920,"2021-09-11 18:59:59 Hora est.
1437204376049815554,1437204376049815554,"2021-09-12 18:59:59 Hora est.
1437566763496087554,1437566763496087554,"2021-09-13 18:59:59 Hora est.
```


Fig 11. Muestra del archivo Overnight.csv. Fuente: autor

El siguiente archivo es forms.py el cual se encarga de 'renderizar' el formulario en la vista cuando la genere flask. El archivo es el siguiente:

```python
from wtforms import Form, StringField, DateField, validators

class KeywordForm(Form):
    keyword = StringField('Keyword')
    language = StringField('Language')
    place = StringField('Place')
    user = StringField('User')
    since = DateField('Since',[
        validators.Required(message = "This field is required!")
        ],
        format='%Y-%m-%d')
    until = DateField('Until',[
        validators.Required(message = "This field is required!")
        ],
        format='%Y-%m-%d',)
```

Fig 12. Archivo forms.py. Fuente: autor

Entonces se tiene una clase llamada KeywordForm que hereda de Form, luego en los atributos se colocan cada uno de los inputs que tendrá el formulario en este caso se tienen seis campos que son keyword, language, place, user, since y until.

Por último, está el archivo main.py que contiene toda la lógica de la aplicación. Primero están las importaciones necesarias luego se realiza la configuración para utilizar flask y twint (como se mostró en pasos anteriores). Luego se definen las siguientes variables globales:

```python
tweets = []
t_data = {}  # dictionary for twitter data
my_json = {"tweets": []}
lang_data = {}  # dictionary for language data
region_data = {}  # dictionary for region data
```

Fig 13. Variables globales. Fuente: autor

Seguido de esto, se definen las rutas para la aplicación en este caso son cuatro, en la figura 14 se muestra el código para la ruta raíz, está se encarga de retornar la renderización del archivo 'index.html'. Luego se configura la ruta '/tweets_template' (Figura 15) que retorna la renderización del archivo 'tweets.html' (este paso es importante para que en la vista, los tuits se refresquen cada vez que se realiza una búsqueda). También se configura la ruta para finalizar una búsqueda '/del' (Figura 16). Por último, configura la ruta más importante '/get' (Figura 17), esta se encarga de extraer los tuits y retornar el análisis de estos.

```python
@app.route('/')
def index():
    keyword_form = forms.KeywordForm()
    return render_template('index.html', form=keyword_form)
```

Fig 14. Ruta raíz

```python
@app.route('/tweets_template')
def tweets_template():
    return render_template('tweets.html')
```

Fig 15. Ruta para renderizar tuits

```python
@app.route('/del', methods=['GET', 'POST'])
def delete():
    twint.output.clean_lists()
    keyword_form = forms.KeywordForm()
    return render_template('index.html', form=keyword_form)
```

Fig 16. Ruta para finalizar búsqueda

```python
@app.route('/get', methods=['GET', 'POST'])
def get():
    query = request.args.get('query')
    lang = request.args.get('lang', 'any')
    place = request.args.get('place', 'any')
    user = request.args.get('user', 'any')
    since = request.args.get('since')
    until = request.args.get('until')
    fetched_tweets = get_tweets(query, lang, place, user, since, until)
    pol = get_polarity(fetched_tweets)
    return pol
```

Fig 17. Ruta para obtener los tuits y su análisis

A continuación, se explicará el funcionamiento de las funciones de las que hace uso la ruta get. En primer lugar la función get_tweets (Figura 18), limpia las variables globales donde se guarda la información de anteriores búsquedas, después obtiene los tuits con la ayuda del objeto perteneciente a twint.Config, al cual se le asignan los parámetros que le son suministrados (keyword, lang, place, user, since y until) además de establecer el límite de tuits que va extraer (c.Count y c.Limit) para este ejemplo diez. En seguida se guardan los tuits contenidos en twint.output.tweets_list en una variable llamada public_tweets. Después se limpia la configuración de twint para realizar nuevas búsquedas posteriormente.

Después, por cada tuit se almacena el texto de este en el arreglo llamado tweets y se añade el tuit a la variable my_json con la siguiente estructura:

- Tweet_Text (Texto del tuit)
- url (Enlace del tuit)
- user_name y screen_name (Nombre de usuario del autor del tuit)
- created_at (Fecha en la que se creó el tuit)
- name (Nombre del autor del tuit)
- profile_img (Imagen de perfil del autor del tuit)
- Polarity (Sentimiento analizado del tuit)

```python
def get_tweets(keyword, lang, place, user, since, until):
    global tweets, t_data, my_json, lang_data, region_data
    tweets.clear()
    lang_data.clear()
    region_data.clear()
    my_json["tweets"].clear()
    c = twint.Config()
    c.Store_object = True
    c.Hide_output = True
    #twint.output.clean_lists()
    c.Search = keyword
    if lang != 'any':
        c.Lang = lang
    if place != 'any':
        c.Near = place
    if user != 'any':
        c.Username = user
    if since != 'any':
        c.Since = since
    if until != 'any':
        c.Until = until
    c.Limit = 10
    c.Count = 10
    twint.run.Search(c)
    public_tweets = twint.output.tweets_list
    c.Lang = None
    c.Near = None
    c.Since = None
    c.Until = None
    c.Username = None
    for tweet in public_tweets:
        tweets.append(tweet.tweet)
        t_data = {
            "Tweet_Text": tweet.tweet,
            "url": tweet.link,
            "user_name": tweet.username,
            "screen_name": tweet.username,
            "created_at": tweet.datestamp,
            "name": tweet.name,
            "profile_image": tweet.profile_image_url,
            "polarity": sentiment(tweet.tweet)
        }
        generate_lang_data(tweet.lang)
        generate_region_data(tweet.near,sentiment(tweet.tweet))
        my_json["tweets"].append(t_data)
    return tweets
```

Fig 18. Función get_tweets

Por último, se va generando la información del lenguaje y de la región con ayuda de las funciones generate_lang_data (Figura 19) y generate_region_data (Figura 20).

```python
def generate_lang_data(lang):
    global lang_data
    if lang in lang_data:
        val = lang_data[lang]
        lang_data[lang] = val + 1
    else:
        lang_data[lang] = 1
```

Fig 19. Función generate_lang_data

Lo que realiza esta función es ir agregando información al diccionario ‘lang_data’ de modo que por cada lenguaje detectado almacena la cantidad de tuits pertenecientes a este lenguaje. Por ejemplo, el diccionario resultante de diez tuits en español, cinco en inglés y 20 en francés es el siguiente:

{‘es’:10, ‘en’:5, ‘fr’:20};

```python
def generate_region_data(region,sentiment):
    global region_data
    if sentiment == "Positive":
        s = 1
    elif sentiment == "Negative":
        s = -1
    else:
        s = 0
    if region in region_data:
        val = region_data[region]
        val[0] = val[0]+1
        val[1] += s
        region_data[region] = val
    else:
        val = [0,0]
        val[0] = 1
        val[1] = s
        region_data[region]= val
```

Fig 20. Función generate_region_data

La función ‘generate_region_data’ va agregando información en el diccionario ‘region_data’ de modo que por cada región en la que se realiza una búsqueda se va almacenando la cantidad de tuits y la suma de la polaridad de los sentimientos para luego en el gráfico de regiones colorear la región de acuerdo a la polaridad promedio en esta región. Por ejemplo, el diccionario resultante de diez tuits con polaridad positiva en Colombia, ocho tuits de los cuales cuatro son positivos y cuatro negativos en Brasil y tres tuits dos con polaridad positiva y uno con polaridad negativa en Alemania es el siguiente:

{‘Colombia’:[10,10], ‘Brazil’:[8,0], ‘Germany’: [3,2]};

La otra función de la que hace uso la ruta ‘/get’, es la función get_polarity (Figura 21), la cual se encarga de asignar en un diccionario la cantidad de tuits que pertenecen a cada polaridad, por ejemplo:

{“Positive”: 10, “Neutral”: 20 , “Negative”: 5}

Después de esto en otro diccionario organiza una estructura tipo JSON donde se encuentra el diccionario de polaridades, el diccionario de tuits, el diccionario de lenguajes y el diccionario de regiones.

```python
def get_polarity(fetched_tweets):
    global my_json, lang_data, region_data

    pos = 0
    neg = 0
    neu = 0

    for tw in fetched_tweets:
        pol = sentiment(tw)
        if pol == 'Positive':
            pos = pos + 1

        if pol == 'Neutral':
            neu = neu + 1

        if pol == 'Negative':
            neg = neg + 1

    # try:
    #     pos_p = (pos / len(fetched_tweets)) * 100
    # except:
    #     pos_p = 0

    # try:
    #     neg_p = (neg / len(fetched_tweets)) * 100
    # except:
    #     neg_p = 0

    # try:
    #     neu_p = (neu / len(fetched_tweets)) * 100
    # except:
    #     neu_p = 0

    pol_dict_data = {"positive": pos, "negative": neg, "neutral": neu}
    sent_dict = {"Sentiment": pol_dict_data, "Tweets": my_json,
                 "Languages": lang_data, "Regions": region_data}
    return sent_dict
```

Fig 21. Función get_polarity

En la parte final del archivo ‘main.py’ se encuentra el ‘main’ que ejecuta la aplicación (Figura 22):

```python
if __name__ == '__main__':
    app.run()
```

Fig 22. Main

Ahora se detallarán los archivos que contienen las carpetas en la raíz, en primer lugar, está la carpeta ‘static’, Flask identifica esta carpeta como los archivos de imágenes, css, js, entre otros. Dentro de esta carpeta se tienen tres subcarpetas como se muestra en la Figura 23.

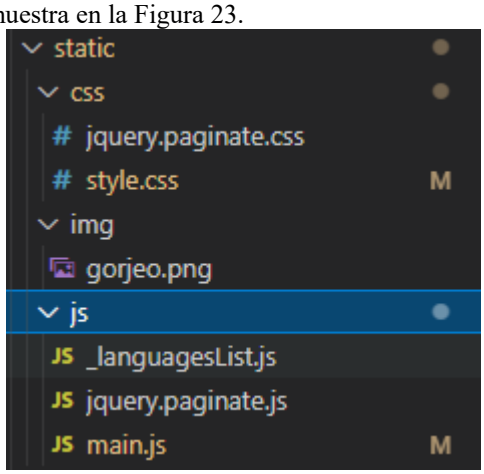

Fig 23. Estructura carpeta static

La carpeta css contiene las hojas de estilo que serán utilizadas por las plantillas html, en este caso el archivo ‘style.css’ es la hoja de estilos principal y el archivo ‘jquery.paginate.css’ corresponde a los estilos de la paginación de los tuits.

La carpeta img contiene las imágenes a utilizar, en este caso solo se tiene una (Figura 24)

Fig 24. Gorjeo.png

La carpeta js contiene los archivos JavaScript que se van a utilizar. En primer lugar, está el archivo ‘_languajesList.js’ (Figura 25), el cual contiene todos los lenguajes en los que se puede realizar una búsqueda (184 exactamente).

```
const languagesList = [
    {
        code: 'aa',
        language: 'Afar'
    },
    {
        code: 'ab',
        language: 'Abkhazian'
    },
    {
        code: 'ae',
        language: 'Avestan'
    },
    {
        code: 'af',
        language: 'Afrikaans'
    },
    {
        code: 'ak',
        language: 'Akan'
    },
    {
        code: 'am',
        language: 'Amharic'
    },
    {
        code: 'an',
        language: 'Aragonese'
    },
    {
        code: 'ar',
        language: 'Arabic'
    },
```

Fig 25. Muestra de lista de lenguajes

Luego está el archivo ‘jquery.paginate.js’ que corresponde al funcionamiento de la paginación de los tuits. Por último, se encuentra el archivo ‘main.js’ que se encargará de la comunicación con el back-end. Inicialmente se importa la lista de lenguajes y los paquetes de Google necesarios para dibujar los gráficos, además de la Api Key necesaria para poder utilizarlos, luego se definen las url para obtener los tuits y finalizar la búsqueda (Figura 26).

```
import { languagesList } from "./_languagesList.js";
google.charts.load("current", {
  packages: ["corechart", "geochart", "calendar"],
  'mapsApiKey': 'AIzaSyAMGlIzu-uy-cWpLeXiBgJb229UzYRMZLM'
});
var url = "http://127.0.0.1:5000/get?query=";
var url_del = "http://127.0.0.1:5000/del";
```

Fig 26. Configuración inicial

El Api Key es necesario para poder utilizar los mapas de Google, para ello es necesario crear una cuenta en Google Cloud Platform que ofrece diferentes servicios y api’s para poderlos integrar en cualquier aplicación. Para generar la llave es necesario crear un proyecto como se muestra en la Figura 27 La documentación de esta herramienta se puede consultar en [20].

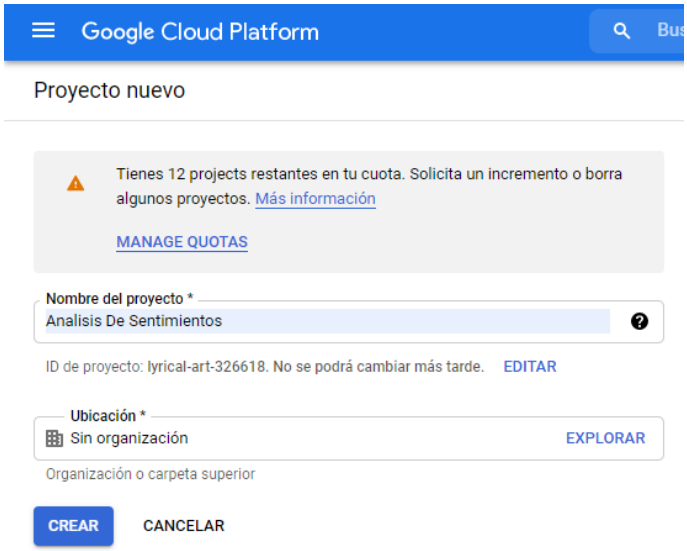

Fig 27. Creación de proyecto en Google Cloud Platform

Luego se definen las funciones que manejarán los eventos de los formularios con ayuda de JQuery. Cuando se hace submit del formulario perteneciente a la búsqueda se obtienen los valores de los inputs y se envía una petición Ajax de la siguiente manera:

```
$.ajax({
  url: u,
  type: "POST",
  success: function (response) {
    $("#for").html("for " + keyword);
    $("#hr").html("<hr/>");
    positive = response["Sentiment"]["positive"];
    negative = response["Sentiment"]["negative"];
    neutral = response["Sentiment"]["neutral"];
    schart(positive, negative, neutral);
    language_chart(response["Languages"]);
    drawRegionsMap(response["Regions"]);
    drawCalendar(response["Tweets"]["tweets"],since_year,until_year);
    aux_tweets = response["Tweets"]["tweets"];
    tweets(response["Tweets"]["tweets"]);
    $(".loader-container").css("display", "none");
  },
});
```

Fig 28. Petición Ajax get

Esta petición se encarga de pedir los tuits en la ruta ‘/get’, y una vez recibida la respuesta dibuja los gráficos, renderiza los tuits y oculta el loader. Por otra parte, al hacer submit del formulario de finalización de la búsqueda se hace otra petición Ajax con la ruta ‘/del’ y se redirige a la raíz, de la siguiente manera:

```
$.ajax({
  url: url_del,
  type: "POST",
  success: function (response) {
    window.location.href = "http://127.0.0.1:5000/"
  },
});
```

Fig 29. Petición Ajax del

Para refrescar la vista de los tuits se hace una petición a la ruta ‘/tweets_template’ como se muestra en la Figura 30. Para dibujar los gráficos se implementan las respectivas funciones para cada uno, los gráficos de torta se realizan con el gráfico de Google ‘Pie Chart’ [21], el gráfico de regiones se realiza

con el gráfico de Google 'Geo Chart' [22] y el gráfico de calendario se realiza con el gráfico de Google 'Calendar Chart' [23].

```
$("#tweets").load("/tweets_template", function () {
```

Fig 30. Petición para refrescar la vista de tuits

Y en último lugar está la carpeta 'templates', Flask identifica esta carpeta como la contenedora de todas las plantillas a renderizar, es decir los archivos con extensión html. Dentro de esta se encuentran tres archivos. Primero el archivo '_macro.html' (Figura 31) el cual se utiliza para la renderización de los inputs de una manera estándar (label e input).

```
{% macro render_field(field) %}
    {{ field.label }}
    {{ field(**kwargs)|safe }}

{% endmacro %}
```

Fig 31. Macro para renderizar inputs

Luego se encuentra el archivo 'tweets.html' (Figura 32) el cual contiene la estructura donde se renderizará la estructura de los tuits, se pone en un archivo aparte del index para poder refrescar esta sección sin tener que recargar la página.

```
<div id="tweet_heading"></div>
<div id="tweet_body"></div>
```

Fig 32. Archivo tweets.html

Por último, está el archivo 'index.html' que contiene la vista principal de la aplicación, en este se realizan los llamados a JQuery 3.4.1 y Bootstrap 4.4.1. En la Figura 33 se muestra como se renderizan los inputs con la ayuda del macro en un formulario.

```
{% from "_macro.html" import render_field %}
<form action="#" id="tweet">
  <div class="form-row">
    <div class="col">
      {{render_field(form.user, class="form-control", placeholder="Enter User", id="user", style="margin-right:1%;")}}
    </div>
    <div class="col">
      {{render_field(form.keyword, class="form-control", placeholder="Enter Keyword", id="keyword", style="margin-right:1%;")}}
    </div>
  </div>
  <div class="form-row">
    <div class="col">
      {{render_field(form.language, class="form-control", placeholder="Enter language", id="language", style="margin-right:1%;",
    </div>
    <div class="col">
      {{render_field(form.since, class="form-control", placeholder="Enter since date", id="since", style="margin-right:1%;")}}
    </div>
  </div>
  <div class="form-row">
    <div class="col">
      {{render_field(form.place, class="form-control", placeholder="Enter Place", id="place", style="margin-right:1%;")}}
    </div>
    <div class="col">
      {{render_field(form.until, class="form-control", placeholder="Enter until date", id="until", style="margin-right:1%;")}}
    </div>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Fig 33. Aplicación del macro en un formulario

VIII. Implementación

En esta sección se pondrá en marcha la ejecución de la aplicación, se mostrarán las diferentes búsquedas que se pueden realizar y los resultados que generan estas. Para poner en marcha la aplicación, se debe situar en la carpeta raíz y en la consola ejecutar el comando 'python main.py', el cual iniciará la aplicación en el puerto 5000 como se puede apreciar en la Figura 34.

```
(twintv2-env) E:\Datos C\Documents\U-V\2021-1\Redes I\Twint\flask-server>python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig 34. Puesta en marcha de la aplicación

Cuando se abre la dirección en el navegador se verá la siguiente interfaz:

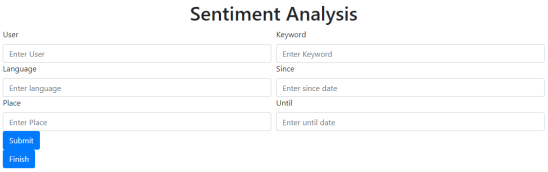

Fig 35. Interfaz inicial

Al realizar la búsqueda mientras se obtiene respuesta se verá el siguiente loader y luego desaparecerá.

Fig 36. Loader

Los tipos de búsquedas que se pueden realizar son:

- Por usuario, con palabras clave, entre determinadas fechas
- Por usuario, sin palabras clave, entre determinadas fechas.
- Por palabras clave, ente determinadas fechas.
- Por lenguaje, con palabras clave, entre determinadas fechas.
- Por país, con palabras clave, entre determinadas fechas (para mayor precisión se recomienda realizar la búsqueda por país con el idioma que se habla en este).

Cuando se realizan búsquedas sin un país especificado el gráfico se mostrará incoloro. Mientras que, si se especifican países, estos se colorearán de acuerdo al sentimiento promedio (El sentimiento se podrá observar pasando el mouse sobre el país).

Fig 37. Mapa sin países dados

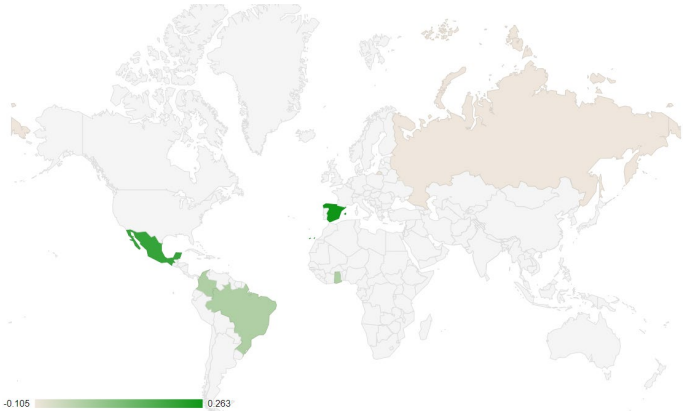

Fig 38. Mapa con países dados

El gráfico de sentimientos muestra la proporción de cada una de las tres polaridades (Positivo, Neutral y Negativo) con relación a la cantidad de tuits. Si se da click en una de las 'porciones de torta' se podrá ver la cantidad de tuits que pertenecen a esta polaridad, su respectivo porcentaje y también se pueden ver los tuits que tienen solo esa polaridad, es decir, ver solo los tuits con polaridad positiva ó neutral ó negativa.

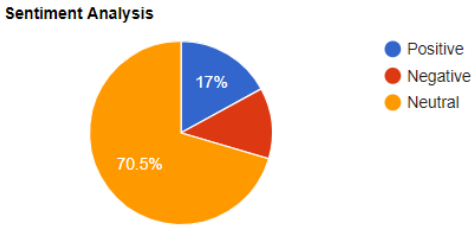

Fig 39. Diagrama de polaridades

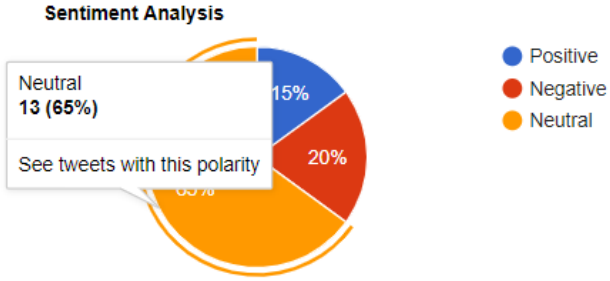

Fig 40. Diagrama de polaridades al dar clic en una porción

El diagrama de lenguajes muestra cada lenguaje con su respectivo porcentaje. Tiene tres comportamientos, el primero cuando no se especifican lenguajes y obtiene gran variedad de tuits en diferentes lenguajes. El segundo comportamiento cuando solo se especifica un lenguaje, de modo que habrá solo un lenguaje en el que están escritos el 100% de los tuits. El tercer comportamiento cuando se especifican n idiomas, de modo que obtendrá tuits de manera proporcional, es decir, si se buscan cien tuits en dos idiomas se obtendran 50 en un idioma y 50 en el otro, que obedece la ecuación (total_tuits/n).

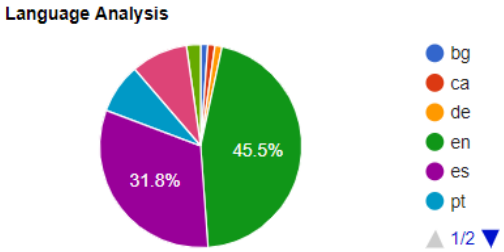

Fig 41. Diagrama de lenguaje sin lenguajes dados

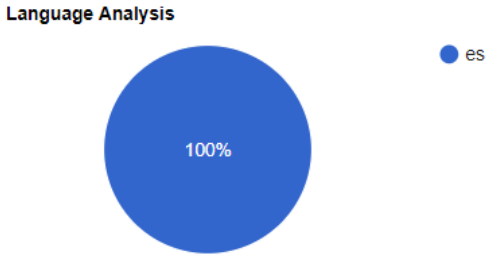

Fig 42. Diagrama de lenguaje con un lenguaje dado

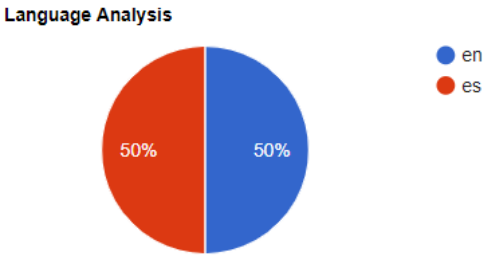

Fig 43. Diagrama de lenguaje con n lenguajes dados

Si la búsqueda se realiza sin usuario, entonces se seleccionan tuits de usuarios aleatorios, en caso contrario solo se seleccionan tiuts del usuario especificado. Esto se puede evidenciar en la sección 'Source Tweets'.

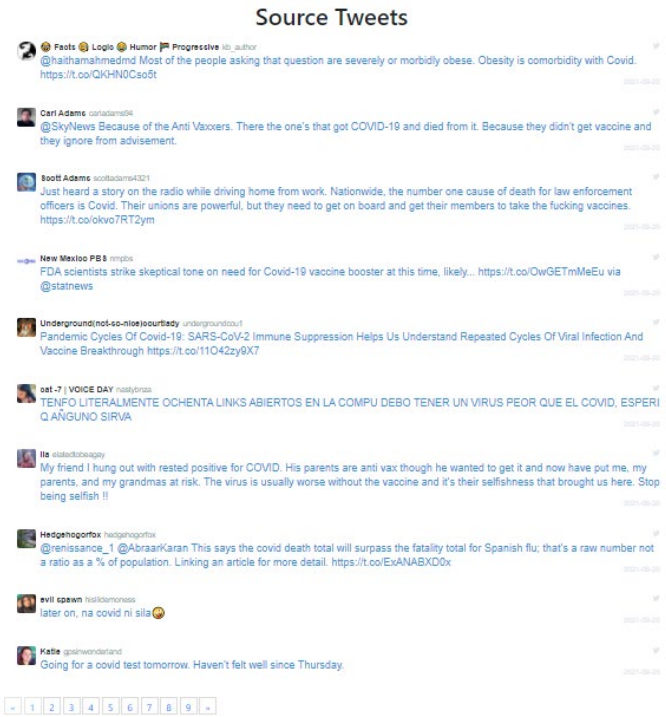

Fig 44. Tuits recolectados de usuarios aleatorios con la palabra clave covid

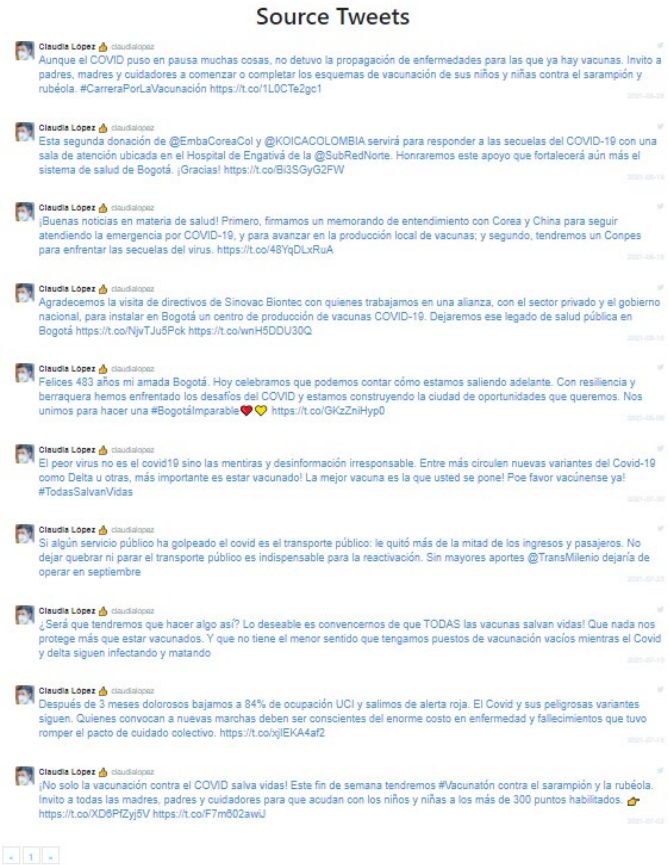

Fig 45. Tuits recolectados de la cuenta @ClaudiaLopez con la palabra clave covid

El gráfico de sentimientos muestra un color por día depeniendo del sentimiento encontrado. Por ejemplo en esta búsqueda realizada al usuario ClaudiaLopez, con las palabras clave Covid y Economía, se puede observar una tendencia neutral en sus tuits (color Marrón) con algunos días con tuits positivos (color Verde) y solo un día en el mes de Agosto del 2020 se evidencia un sentimiento negativo (colo Rojo).

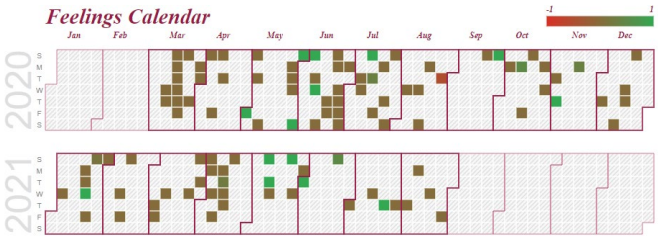

Fig 46. Diagrama de sentimientos

IX. Limitaciones

Algunas de las limitaciones para el desarrollo e implementación del proyecto fueron las siguientes:

- En un principio se utilizaría la API oficial de Twitter llamada Tweepy, sin embargo, esta API solo permite realizar búsquedas entre la fecha actual y siete días atrás, por lo que se optó por la librería Twint que realiza peticiones a Twitter sin restricciones.
- La Api Key dada por Google Cloud Platform solo funciona por 30 días desde su generación en su versión de prueba, por lo que para seguir utilizando las funciones completas de la aplicación se debe comprar la licencia para usar los servicios, o en otro caso, crear otro proyecto para generar otra Api Key durante 30 días de prueba. Cabe advertir que solo se pueden crear 12 proyectos en la versión de prueba gratuita.
- Por último, aunque la librería Twint nos permite realizar peticiones a Twitter sin restricciones, si se realizan varias peticiones en un periodo corto de tiempo puede que se obtenga el siguiente error:

```
[!] No more data! Scraping will stop now.
found 0 deleted tweets in this search.
[+] Finished: Successfully collected 0 Tweets from @ClaudiaLopez.
```

Fig 47. Error de bloqueo ip

Esto significa que se ha bloqueado la ip desde donde se realizan las peticiones, para superar este veto, se debe esperar al menos 7 minutos, antes de volver a realizar una petición, lo que hace algo lento el proceso de análisis algunas veces.

X. Conclusiones

En primer lugar, se ha realizado una aplicación con una interfaz amable para el usuario que le permite realizar búsquedas sobre Twitter. Los tuits obtenidos son analizados y almacenados de manera temporal, de modo que el usuario puede apreciar los distintos resultados de polaridad que obtiene la búsqueda. Esto permite valorar qué opinión tiene el grueso de usuarios de Twitter acerca del tema buscado, o la opinión de un usuario respecto a un tema en específico, o la evolución sentimental que expresa un usuario en sus tuits, además de

poder filtrar por países y lenguajes para realizar comparaciones entre estos.

En cuanto a la herramienta utilizada para el despliegue, Flask es una gran herramienta de trabajo para desarrollar aplicaciones web de manera efectiva, rápida y sencilla. Se adaptó de manera perfecta a la estructura MVC que se planteó, reduciendo notablemente el tiempo de codificación para la comunicación entre vista y modelo del proyecto.

La librería para obtener los tuits Twint, resultó la mejor opción después de descartar Twepy (librería oficial de Twitter para la extracción de datos), debido a sus restricciones en cuanto a cantidad de tuits, y tiempo de búsqueda (un rango de siete días), a diferencia de Twint que no tiene restricciones y también se comunica directamente con la API de Twitter, sin embargo, si se realizan demasiadas búsquedas, puede que la API de Twitter bloquee la ip dónde se originan las peticiones por un periodo de tiempo, durante la fase de pruebas el máximo tiempo por el que se bloqueó la ip fue de siete minutos.

Por último, como trabajo para realizar a futuro, se debería implementar una base de datos donde se pueda almacenar la información esencial de los tuits extraídos en una búsqueda, primero para no tener que realizar una búsqueda igual para recuperar los mismos tuits y segundo para realizar un análisis más amplio ya que almacenando los tuits de manera permanente se pueden realizar análisis más complejos.

XI. Apéndice

[1] Enlace del repositorio que contiene el proyecto: https://github.com/ChristianGalindo10/ProyectoRedes1

Referencias

[1] Peteet JR. COVID-19 Anxiety. J Relig Health. 2020 Oct; 59(5):2203-2204. doi: 10.1007/s10943-020-01041-4. PMID: 32415426; PMCID: PMC7227179.

[2] *Sher L. COVID-19, anxiety, sleep disturbances and suicide. Sleep Med. 2020 Jun; 70:124. doi: 10.1016/j.sleep.2020.04.019. Epub 2020 Apr 25. PMID: 32408252; PMCID: PMC7195057.*

[3] J. N. Franco-Riquelme, A. Bello-Garcia and J. Ordieres-Meré, "Indicator Proposal for Measuring Regional Political Support for the Electoral Process on Twitter: The Case of Spain's 2015 and 2016 General Elections," in IEEE Access, vol. 7, pp. 62545-62560, 2019, doi: 10.1109/ACCESS.2019.2917398.

[4] P. Wang et al., "Classification of Proactive Personality: Text Mining Based on Weibo Text and Short-Answer Questions Text," in IEEE Access, vol. 8, pp. 97370-97382, 2020, doi: 10.1109/ACCESS.2020.2995905.

[5] T. Wang, K. Lu, K. P. Chow and Q. Zhu, "COVID-19 Sensing: Negative Sentiment Analysis on Social Media in China via BERT Model," in IEEE Access, vol. 8, pp. 138162-138169, 2020, doi: 10.1109/ACCESS.2020.3012595.

[6] F. Yin, Y. Wang, J. Liu and L. Lin, "The Construction of Sentiment Lexicon Based on Context-Dependent Part-of-Speech Chunks for Semantic Disambiguation," in IEEE Access, vol. 8, pp. 63359-63367, 2020, doi: 10.1109/ACCESS.2020.2984284.

[7] K. Cheng, Y. Yue and Z. Song, "Sentiment Classification Based on Part-of-Speech and Self-Attention Mechanism," in IEEE Access, vol. 8, pp. 16387-16396, 2020, doi: 10.1109/ACCESS.2020.2967103.

[8] K. Abdalgader and A. A. Shibli, "Experimental Results on Customer Reviews Using Lexicon-Based Word Polarity Identification Method," in IEEE Access, vol. 8, pp. 179955-179969, 2020, doi: 10.1109/ACCESS.2020.3028260.

[9] Y. Kim and M. Kim, "'A Wisdom of Crowds': Social Media Mining for Soccer Match Analysis," in IEEE Access, vol. 7, pp. 52634-52639, 2019, doi: 10.1109/ACCESS.2019.2912009.

[10] A. Karami, M. Lundy, F. Webb and Y. K. Dwivedi, "Twitter and Research: A Systematic Literature Review Through Text Mining," in IEEE Access, vol. 8, pp. 67698-67717, 2020, doi: 10.1109/ACCESS.2020.2983656.

[11] J. A. Diaz-Garcia, M. D. Ruiz and M. J. Martin-Bautista, "Non-Query-Based Pattern Mining and Sentiment Analysis for Massive Microblogging Online Texts," in IEEE Access, vol. 8, pp. 78166-78182, 2020, doi: 10.1109/ACCESS.2020.2990461.

[12] D. Liu et al., "Suicidal Ideation Cause Extraction From Social Texts," in IEEE Access, vol. 8, pp. 169333-169351, 2020, doi: 10.1109/ACCESS.2020.3019491.

[13] Digital Guide IONOS. (2019, March 11). *El modelo en cascada: desarrollo secuencial de software.* [Online]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/

[14] Pallets Projects. *Flask Documentation.* [Online]. Available: https://palletsprojects.com/p/flask/

[15] Manual Web. *Request Flask*. [Online]. Available: http://www.manualweb.net/flask/request-flask/

[16] Read the Docs. *Flask-CORS. [Online]*. Available: https://flask-cors.readthedocs.io/en/latest/

[17] Read the Docs. *TextBlob: Simplified Text Processing*. [Online]. Available: thttps://textblob.readthedocs.io/en/dev/

[18] Unypython. (2018, Dec 10). *ANÁLISIS DE SENTIMIENTOS CON TEXTBLOB Y VADER EN PYTHON*. [Online]. Available: https://unipython.com/analisis-de-sentimientos-con-textblob-y-vader/

[19] OSINT team (2021, March 2). *TWINT - Twitter Intelligence Tool*. [Online]. Available: https://github.com/twintproject/twint

[20] Google Charts. *Comienza a usar Google Cloud*. [Online]. Available: https://cloud.google.com/docs

[21] Google Charts. *Pie Chart.* [Online]. Available: https://developers.google.com/chart/interactive/docs/gallery/piechart

[22] Google Charts. *Geo Chart.* [Online]. Available: https://developers.google.com/chart/interactive/docs/gallery/geochart

[23] Google Charts. *Calendar Chart.* [Online]. Available: https://developers.google.com/chart/interactive/docs/gallery/calendar