ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

MODELOS ESTADÍSTICOS PARA LA DETECCIÓN DE PATRONES EN MEDIO AMBIENTE Y FINANZAS

ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES SAR DE DERRAMES DE PETRÓLEO USANDO LA PLATAFORMA *OPEN ACCES HUB* Y LA PLATAFORMA SNAP UTILIZANDO TWITTER COMO FUENTE DE INFORMACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

ADRIÁN JOSUÉ ENRÍQUEZ PUETATE

adrian.enriquez@epn.edu.ec

DIRECTOR: PH. D. MIGUEL ALFONSO FLORES SÁNCHEZ

miguel.flores@epn.edu.ec

DMQ, FEBRERO 2022

CERTIFICACIONES

Yo, ADRIÁN JOSUÉ ENRÍQUEZ PUETATE, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Adrián Josué Enríquez Puetate

Certifico que el presente trabajo de integración curricular fue desarrollado por Adrián Josué Enríquez Puetate, bajo mi supervisión.

Ph. D. Miguel Alfonso Flores Sánchez
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Adrián Josué Enríquez Puetate

Ph. D. Miguel Alfonso Flores Sánchez

DEDICATORIA

A mis padres, por haberme acompañado en este largo y duro trayecto.

AGRADECIMIENTOS

A mis padres, Patricio y Mery, por ser mis mentores desde el primer día de mi vida, por guiarme en el camino del bien mediante los valores.
A mis abuelitos, quienes con sus consejos y sus palabras de aliento no me dejaron rendir.
A mis tíos y primos, por apoyarme incondicionalmente. En especial a mi tío Pablo, quien es como mi segundo padre; a mi tía Mayra, quien es como mi segunda madre; a mi primo Luis, quien me acompaño en mis primeros años de la universidad y siempre estuvo ahí cuando lo necesitaba.
A Viviana, por la confianza, paciencia y apoyo en el desarrollo de este trabajo de integración curricular.
A Miguel Flores, por depositar su confianza en mi.

RESUMEN

Los derrames de petróleo han sido uno de los causantes de la pérdida de flora y fauna en todo el mundo. Además, es considerado como uno de los problemas más grandes en cuanto a medio ambiente se refiere. Por lo que en este componente se presenta un método para recopilar datos acerca de derrames de hidrocarburos mediante el uso de los reportes que las personas realizan por medio de la red social Twitter. Para obtener esta información se hace uso de la API de Twitter y el *web scraping*. Luego, usando la información se puede obtener imágenes satelitales a travez de la plataforma *Copernicus Open Acces Hub*, las cuales pueden ser procesadas y clasificadas dependiendo si existe o no un derrame de petróleo en ellas. Y así disponer de un método para crear una base de datos con un número considerable de imágenes, esto ayudará a futuras investigaciones a crear modelos de clasificación usando algún algoritmo de aprendizaje.

Palabras clave: Petróleo, derrame, *web scraping*, API, SAR, imagen.

ABSTRACT

Oil spills have been one of the causes of the loss of flora and fauna around the world. In addition, it is considered one of the biggest environmental problems. Therefore, this component presents a method to collect data about oil spills through the use of the reports that people make through the social network Twitter. In order to obtain this Twitter API and web scraping are used to obtain this information. Then, using the information, satellite images can be obtained through the Copernicus Open Acces Hub platform, which can be processed and classified depending on whether or not there is an oil spill on them.

Keywords: Oil, spill, web scraping, API, SAR, image.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

La protección del medio ambiente es un tema importante y de creciente preocupación pública. La contaminación del medio marino causada por los derrames de petróleo es de gran preocupación para los países costeros debido a sus impactos ecológicos, ambientales y socioeconómicos. Todos los estudios que se han realizado apuntan a que la detección y vigilancia de derrames de hidrocarburos es la gestión más importante para lidiar con este inconveniente [9]. Un derrame de petróleo es una amenaza ambiental que tiene efectos adversos en las masas de agua, la tierra y el aire [2] son causadas principalmente por accidentes petroleros, barcos y oleoductos en los que se vierte petróleo crudo, gasolina, combustible y subproductos del petróleo en el agua.

La red social *Twitter* ha sido usada en los últimos años como una fuente para divulgar y consumir información, parte de esta información consta de los reportes que realizan los individuos acerca de derrames de petróleo, así, mediante el uso de la *API* que esta red social brinda a desarrolladores se puede extraer los lugares en los que se realizaron estos reportes. Que junto con el *web scraping* se crea un conjunto de lugares en los cuales ocurrieron posibles derrames de petróleo[1].

El radar de apertura sintética (SAR) es un tipo de radar que se utiliza para crear imágenes bidimensionales o tridimensionales de objetos [5]. El

[1]Son posibles derrames ya que *Twitter* dispone de una gran cantidad de usuarios, los cuales pueden publicar cualquier tipo de información ya sea verdadera o falsa

radar envía pulsos electromagnéticos a la tierra con el objetivo de grabar su retorno una vez reflejado por la superficie y tiene una alta capacidad de trabajo en todo tipo de condiciones meteorológicas como también a cualquier hora del día. Este radar se ha considerado como una de las herramientas más potentes para la detección de derrames de petróleo en la superficie del mar. Sin embargo, en las imágenes SAR aparecen con frecuencia elementos similares que limitan el uso operativo del SAR para detectar el petróleo derramado.

La Agencia Espacial Europea (ESA) se encarga de distribuir las imágenes SAR obtenidas en las misiones del Sentinel-1 y se puede extraer a través de varias plataformas que la Agencia ofrece. Para el desarrollo de este componente se usa la plataforma *Open Acces Hub* que permite recopilar las imágenes SAR conociendo las coordenadas geográficas del lugar exacto donde ocurrió un posible derrame de petróleo como también la fecha. Luego, para el procesamiento se usa el módulo *snappy* que se encuentra asociado al lenguaje de programación Python y pertenece a la plataforma SNAP, la cual ha sido desarrollada para facilitar el uso, la visualización y procesamiento de imágenes SAR [1].

El hecho de crear una base de datos que indique derrames de petróleo incentiva a la creación de modelos de aprendizaje supervisado. Pues el desarrollo de sistemas automatizados o semiautomatizados para la detección de derrames de hidrocarburos es un tema de varios esfuerzos reportados en la literatura [4]. Puesto que el uso de las imágenes SAR para detectar derrames de petróleo no es del 100 % certera, se han investigado varios enfoques para clasificar este tipo de imágenes utilizando clasificadores de aprendizaje profundo y redes neuronales.

Los derrames de hidrocarburos en el mar producen problemas de gran impacto ambiental, es por esta razón que en este trabajo de integración curricular se proporciona las bases teóricas para la recopilación de un conjunto de imágenes SAR que sirve de ayuda para desarrollar modelos de clasificación para identificar derrames de petróleo producidos en el mar de manera más eficiente. Que posteriormente servirá para crear aplicaciones para que cualquier organización ambiental pueda ser advertida de posibles derrames de hidrocarburos y así reaccionar de manera inmediata evitando grandes pérdidas ya sea en flora y en fauna.

1.1. Objetivo general

Crear una base de datos mediante la recopilación y procesamiento de imágenes SAR tomadas en lugares donde se realizaron reportes de derrames de petróleo para generar una fuente de consulta de derrames ocurridos.

1.2. Objetivos específicos

1. Extraer el conjunto de lugares en los que ocurrieron posibles derrames de petróleo mediante el análisis de publicaciones realizadas en Twitter.

2. Recopilar el conjunto de imágenes SAR para la construcción de la base de datos, en este caso imágenes tomadas por el satélite Sentinel-1 distribuidas por la Agencia Espacial Europea (ESA) a través de *Copernicus Open Acces Hub*.

3. Realizar el preprocesamiento del conjunto de datos mediante técnicas de tratamiento de imágenes satelitales para identificar y etiquetar instancias de derrames de petróleo, objetos sospechosos y otros objetos.

1.3. Alcance

En este trabajo de integración curricular, se pretende recopilar el conjunto de datos usando técnicas de recopilación como lo es el *web scraping*, creando así una base de datos en la cual se especifique la ubicación de reportes de derrames de petróleo que ocurrieron entre los años de 2018 a 2020.

Lo anterior se realiza con el fin de obtener los lugares en los cuales ocurrieron posibles derrames de petróleo, y así poder conseguir las coordenadas geográficas, las cuales son necesarias para recopilar las imágenes tomadas por el satélite Sentinel-1 distribuidas por la Agencia Espacial Europea (ESA) a través de *Copernicus Open Acces Hub*.

Luego de haber recopilado las imágenes satelitales, se hará uso de la plataforma SNAP, la cual provee un módulo llamado *snappy*, el cual se lo puede usar en el lenguaje de programación *Python*. Esto con el fin de procesar y etiquetar las imágenes que presentan derrames de petróleo como también las que no presentan derrames.

Con todo lo anterior se cumple el objetivo general. Permitiendo que todo lo desarrollado en el componente se pueda usar para desarrollar un modelo de clasificación que proporcione información necesaria para la protección ambiental.

1.4. Marco teórico

En esta sección se explicará a detalle las bases teóricas de las herramientas que se va a usar para alcanzar los objetivos planteados, entre ellas están: el *web scraping*, la *API* de la red social *Twitter* y las imágenes SAR.

1.4.1. El *Web Scraping*

El *Web Scraping* o también conocido en español como raspado web, es una técnica utilizada mediante programas de software con la cual podemos recopilar información de cualquier sitio web, cuya información sea de conocimiento público, como por ejemplo: noticias, tablas de clasificación, rankings, etc. El Web Scraping es la recolección automática de información de los sitios web [3], es decir, se diseña un algoritmo con el cual podamos extraer información de cierta página web de interés sin la intervención de un ser humano.

Estructura de una página web

Para comprender de mejor manera el funcionamiento del *web scraping* se debe entender la estructura de una página web. Las páginas web están conformadas por “elementos”, que son unidades con significado, como: títulos, parráfos, listas, tablas, imágenes, enlaces, etc [6].

Los elementos de una página web antes mencionados se representan

usando etiquetas del lenguaje HTML[2]. A continuación se presenta algunos ejemplos de las etiquetas más usadas:

```
1 <h1>Título h1</h1>
2 <p>párrafo</p>
3 <a href="dirección de enlace">Enlace</a>
4 <img src="casa.jpg">
```

Figura 1.1: Ejemplo de etiquetas de HTML

En la línea 1 se tiene la etiqueta *<h1>...</h1>* que sirve para escribir títulos, en la línea 2 se tiene la etiqueta * ... * que sirve para redactar párrafos de texto, en la línea 3 se tiene la etiqueta *<a>...* que representa un enlace y el atributo *href* indica la dirección del enlace, y la línea 4 tiene la etiqueta *<img>* que ayuda a colocar imágenes y el atributo *src* indica la dirección donde se encuentra la imagen.

Según López [6], una página web es una secuencia anidada de elementos, como se muestra en el siguiente ejemplo:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Información importante</title>
5  </head>
6  <body>
7      <h1>Información importante</h1>
8      <p>Se avisa que el próximo viernes serán entregados los
9      reconocimientos a los estudiantes mejor aplicados</p>
10 </body>
11 </html>
```

Figura 1.2: Ejemplo de una página web dada por López [6].

La técnica de *web scraping* consiste en acceder a todos los elementos de la página web de la cual se quiere obtener información. Para ello, se puede usar el lenguaje XPath, que sirve precisamente para indicar uno o varios elementos en particular, incluyendo sus atributos y su contenido [6].

[2]Lenguaje de marcado comúnmente usado para la elaboración de páginas web.

Entre las expresiones más usadas en el lenguaje XPath según López [6] son:

- *//p*: Selecciona todos los elementos *p* de una página;
- *//img/@src*: Selecciona los atributos *src* de todos los elementos *img*;
- *//a/text()*: Selecciona el contenido de todos los elementos *a* (enlaces);
- *//div/span*: Selecciona todos los elementos *span* que se encuentran dentro de un elemento *div*;
- *//th* [@class="centered"]: Selecciona todos los elementos *th* que tengan el atributo *class="centered"*.

Procedimiento

Existen varios procedimientos para realizar *web scraping* y dependiendo de la página web se usará un procedimiento u otro o varios. El que se usará en este componente será el siguiente:

1. Revisar el código fuente de la página web y determinar la forma en la que se encuentran organizados sus elementos para definir las rutas XPath.
2. Crear un programa en Python para descargar el contenido de la página y que extraiga los datos según las rutas XPath definidas anteriormente.
3. Almacenar y procesar los datos.

Para crear el programa de descarga del contenido de la página web en Python se usará las librerías *BeautifulSoup* y *requests* cuyo funcionamiento se dará a conocer en el capítulo (2).

La *API* de *Twitter*

La sigla API *"application programming interface"* es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación

orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software [3]. Ante el temor de que ciertos datos sean robados por **scrapers**, algunas empresas ofrecen el servicio de API's para que los usuarios interesados en obtener información lo pueda hacer mediante la API.

La red social Twitter es una de estas empresas que pone a disposición una API para que las personas interesadas puedan extraer información acerca de los usuarios que están registradas en la red social, como también la información que estas publican, como pueden ser noticias, comentarios, encuestas, etc.

Esta API permitirá obtener acceso a las noticias publicadas respecto a los reportes de derrames de petróleo, permitiendo construir una primera base de datos y así cumplir con los objetivos.

Para el uso de esta API, se usará el lenguaje de programación de Python en conjunto con la librería *tweepy*, cuyo funcionamiento se explicará mejor en el capitulo (2). En el caso de que se desee usar la API en algún otro lenguaje se puede usar las respectivas librerías y paquetes que la API ofrece dependiendo de cada lenguaje de programación. En el caso de usar el lenguaje R se tiene el paquete *RTwitterV2*.

1.4.2. Teoría de las imágenes SAR

El Radar de Apertura Sintética (SAR) logra obtener imágenes de alta definición para teledetección mediante el movimiento de un radar para crear el efecto de una gran antena [12]. Esto ayuda para realizar estudios meteorológicos, ambientales, etc. Para este componente se usará para la detección de derrames de petróleo en el mar.

Geometría del SAR

SAR utiliza solo una antena física para recolectar señales que se reflejan desde la ubicación objetivo a diferentes posiciones y tiempos [12]. Es decir, el Radar de Apertura Sintética envía señales al punto objetivo y analiza las señales que se reflejan creando una imagen a la cual se tendrá que estudiar sus características.

En la siguiente figura se puede apreciar de mejor manera como fun-

ciona un SAR,

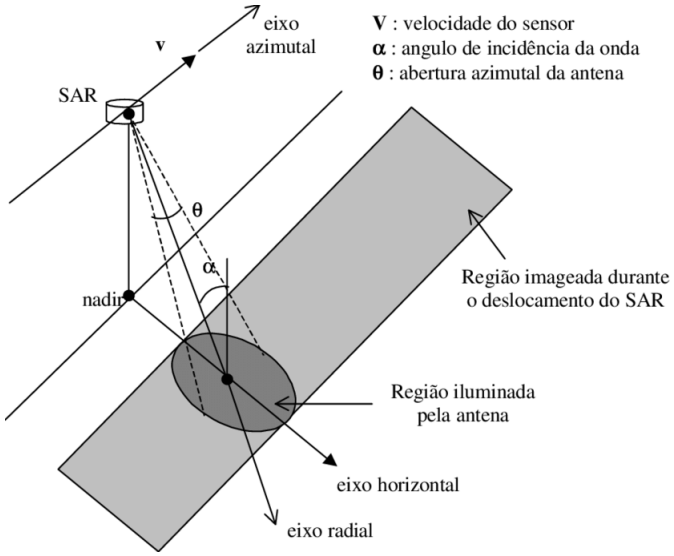


Figura 1.3: Geometría de SAR.

en la cual se pueden distinguir los siguientes elementos:

- Dirección acimutal: dirección en la cual se está moviendo el sensor,
- v: velocidad del sensor,
- θ: apertura acimutal de la antena,
- α: ángulo de inclinación de la señal reflejada respecto a la señal enviada por el SAR,
- Región iluminada por la antena,
- Región de la imagen obtenida tras el desplazmiento del SAR,

Se debe tener en cuenta que la señal reflejada en la superficie debe poder distinguirse de cualquier ruido que exista alrededor. Existen varios tipos de señales, en este componente se enfocará más en las señales de pulso lineal FM, pues es la más común [12].

La señal reflejada por pulsos desde un solo punto objetivo después de la demodulación viene dada por la ecuación [12]

$$s_0(\tau,\eta) = w_r\left(\tau - 2\frac{R(\eta)}{c}\right)\cdot w_a(\eta-\eta_c) \\ \cdot \exp\left\{-j4\pi f_0\frac{R(\eta)}{c}\right\}\cdot\exp\left\{j\pi K_r\left(\tau-2\frac{R(\eta)}{c}\right)^2\right\} \tag{1.1}$$

donde:

- τ: tiempo rápido a lo largo de la dirección del rango,
- η: tiempo lento a lo largo de la dirección acimutal,
- w_r: sobre de rango,
- w_a: sobre de acimutal,
- $R(\eta)$: distancia de alcance instantánea,
- c: velocidad de la luz,
- η_c: tiempo de cruce del centro del haz,
- K_r: tasa de pulso,
- R_0: alcance de la aproximación más cercana,
- V_r: velocidad efectiva del radar

La generación de una imagen SAR corresponde al proceso de centrarse en cada punto objetivo mediante la ponderación, el desplazamiento de fase y la suma de los historiales de fase de las respuestas[12].

Imagen SAR vista computacionalmente

Una imagen SAR puede ser vista computacionalmente como una matriz de píxeles $ND \in \mathbb{R}^{m\times n}$ de la siguiente forma:

$$ND = \begin{pmatrix} ND_{0,0} & ND_{0,1} & \cdots & ND_{0,n-1} \\ ND_{1,0} & ND_{1,1} & \cdots & ND_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ ND_{m-1,0} & ND_{m-1,1} & \cdots & ND_{m-1,n-1} \end{pmatrix} \tag{1.2}$$

las entradas de la matriz de píxeles en una imagen monocramática (niveles de gris) típica son del orden 28 o 256 niveles de grises, de tal manera que, $ND_{i,j}$ puede estar en el rango de 0 a 255, donde 0 representa el negro y 255 el blanco, con lo cual se puede representar como una cadena de caracteres en la mayor parte de lenguajes de programación [10].

Procesamiento de imágen SAR

Para procesar las imágenes SAR en este componente se seguirán los siguientes pasos:

1. Leer la imagen SAR,
2. Sacar un subconjunto de la imagen SAR,
3. Calibración radiométrica,
4. Filtro espectral,

A continuación se explica cada uno de los pasos a partir del segundo.

Subconjunto de una imagen SAR (*Subset*)

Por lo general, una imagen SAR contiene mayor información de la que se necesita, por tanto, se extrae un *subset* de la imagen. Esto ayuda a reducir el tiempo de procesamiento.

Calibración radiométrica de imagen SAR

El objetivo de la calibración radiométrica de la imagen SAR es proveer una imagen, en la cual, cada valor de los píxeles puedan directamente relacionarse para la retrodispersión del radar [1]. La calibración radiométrica para imágenes satelitales en el caso de que algún pixel se haya perdido se la realiza mediante la estimación de ese pixel, esta estimación se la tomaría como la media de los valores del mismo pixel en las líneas anterior y posterior [11].

$$\widehat{ND_{i,j}} = round\left(\frac{ND_{i-1,j} + ND_{i+1,j}}{2}\right) \tag{1.3}$$

donde $round$ indica el redondeo al número más cercano y $ND_{i,j}$ representa el valor del pixel i, j.

Filtro espectral

Al momento de obtener una imagen satelital, pueden existir varios elementos en la imagen que provoquen perdida de visibilidad, es decir, provocan que la imagen no sea muy clara. A estos elementos se les conoce como ruido espectral que se produce por interferencias aleatorias que ocasiona el entorno.

El filtro espectral consiste en reducir este ruido espectral mediante una reducción de la varianza de la dispersión espectral como también con la mejora del coeficiente de dispersión no espectral.

Para la reducción de la varianza de la dispersión espectral existen varias posibilidades, entre ellas están: el filtro de la media, filtro de la media ponderada, filtro de la mediana, filtros adaptativos, filtros gaussianos [11]. A continuación se detalla en que consiste los filtros gaussianos ya que entre ellos está el filtro *Lee Sigma*, el cual es usado para el desarrollo de este componente.

Los filtros gaussianos simulan una distribución gaussiana bivariante, en la cual, el valor máximo aparece en el pixel central y disminuye a los extremos tanto más rápido cuanto menos sea el parámetro de la desviación típica s [11]. La ventaja de usar este filtro es que se puede eliminar la mayor parte del ruido espectral de una imagen, sin embargo, empaña la imagen ocasionando pérdida de los detalles más finos [10]. Esto debido a que los píxeles más lejanos del píxel central tienen una influencia insignificante.

El filtro *Lee Sigma* utiliza la distribución estadística de los valores de cada píxel (ND) para estimar el valor del píxel de interés [8]. El filtro asume una distribución gaussiana para el ruido en la imagen y que la media y varianza del píxel de interés son el promedio y varianza de todos los píxeles [8].

El filtro *Lee Sigma* viene dada por la expresión:

$$ND_{out} = \bar{P} + k\left(ND_{in} - \bar{P}\right) \tag{1.4}$$

donde:

- $\bar{P}$: promedio de píxeles.
- $k = \dfrac{Var(x)}{\bar{P}^2\sigma^2 + Var(x)}$
- $Var(x) = \dfrac{s^2_{ND} + (\bar{ND})^2)}{\sigma^2 + 1} - (\bar{ND})^2$
- σ : parámetro sigma para aplicar el filtro.

Capítulo 2

Metodología

En este capítulo se detalla todo lo relacionado al procedimiento utilizado para el desarrollo del componente. El cual se lo indica de manera resumida a continuación:

Se empieza explicando el proceso de extracción de datos de la red social Twitter, en especial, la información acerca de reportes de derrames de petróleo que se utilizará en este trabajo. Posterior a esto, se da a conocer como se realiza el procesamiento de toda la información antes obtenida. Luego, se explica como se obtiene las coordenadas geográficas correspondientes a la ubicación donde se realizan los reportes. Finalmente, se da a entender como usar las coordenadas para la obtención de las imágenes SAR en la plataforma Copernicus, como también se explica el procesamiento de estas imágenes.

2.1. Recopilación de reportes de derrames de petróleo

Para crear un modelo para la **Identificación de derrames de petróleo en el mar con base en algoritmos de aprendizaje supervisado utilizando imágenes SAR** se necesita de una base de datos con información de los reportes de derrames de petróleo registrados en el Mar Caribe y Golfo de México, en un periodo del 28 de septiembre de 2018 hasta el 31 de

octubre de 2020, esta base constará de las siguientes variables:

- **País:** País de donde se realizó el reporte de derrame.
- **Fecha:** Día, mes y año en el que se hizo el reporte de derrame.
- **Hora:** Hora en la que se realizó el reporte de derrame.
- **Latitud:** Coordenada geográficas de latitud donde se realizó el reporte de derrame.
- **Longitud:** Coordenadas geográficas de longitud donde se realizó el reporte de derrame.
- **Imagen_SAR:** Imagen satelital tomada en la ubicación donde se hizo el reporte de derrame.
- **Reporte:** Variable indicadora, 1 si hubo derrame y 0 si no lo hubo.

2.1.1. Uso de la red social Twitter para la extracción de datos

Para la recopilación de datos se usará la red social Twitter, la cual ha venido siendo uno de los más importantes medios tecnológicos para la publicación de noticias alrededor del mundo. Con lo cual, se puede extraer información publicada acerca de derrames de petróleo.

Para extraer de manera rápida la información necesaria, se emplea la API de twitter, la cual es una herramienta que ofrece Twitter para poder extraer información de esta red social. Para el uso de esta API se necesita tener una cuenta de Twitter y una cuenta de desarrollador, las cuales se pueden crear en el siguiente enlace: https://developer.twitter.com/en/apply-for-access.

Una vez que ya se tiene la cuenta de desarrollador se procede a crear un nuevo proyecto en la interfaz de la API y una app para luego generar las credenciales[1] que permitirán hacer uso de la API en algún lenguaje de programación, en nuestro caso se usará Python.

[1] se conocen como *keys* y *Tokens*

Después de haber generado las credenciales necesarias, se procede a copiarlas en un archivo con nombre *credentials.py* de la siguiente manera:

```
API_KEY = "AQCP5VbFWeBIST4dLzn74gIFw"

API_SECRET_KEY = "rUjQfPrHx82TMoD8Wv3DDJqiQ7HOrQOKUBz6TbElcPYRYbM2ZI"

ACCES_TOKEN= "1476723096266620929-ic5218M92m764fkcZDu5i8v0rUhReF"

ACCES_TOKEN_SECRET= "J2F5nZrxm5Ut3bUUgrILaTYrjAtugtJQyAyHNIOsW19cu"
```

Figura 2.1: Credenciales de la API

Luego, en el menu de la interfaz de desarrollador de la API de Twitter se despliega el submenu de *productos* y luego ingresar *Dev Enviroments* que se encuentra en *Premium*. Finalmente configuramos un ambiente con *Sandbox, Search twetts: full archive* seleccionando la app antes creada y escribiendo un nombre al ambiente.

Lo anterior se lo realiza con la finalidad de obtener datos historicos de la API, ya que si no se activa el ambiente de *Sandbox*, la API solo ofrecerá datos con solo 30 días de haber sido publicados.

Ahora ya se tiene todo lo necesario para empezar a usa la API en conjunto con el lenguaje de programación Python, en el cual necesitarémos de los módulos: *tweepy* y *credentials*. Donde *tweepy* se lo puede instalar escribiendo el comando *pip install tweepy* en cualquier terminal que se tenga instalado Python, y *credentials* es el archivo que se creó anteriormente para guardar las credenciales de la API.

El módulo *tweepy* permite acceder a todas las funcionalidades que brinda la API a través del lenguaje de Python, se puede realizar búsquedas de todas las publicaciones de Twitter, ver los usuarios con toda su información publicada en la red social, etc.

Ahora, se comienza a importar los módulos antes mencionados, el módulo *pandas* y el módulo *json* mediante las siguientes líneas de código:

```
import tweepy
import credentials
import pandas as pd
import json
```

Figura 2.2: Importación de los módulos *tweepy* y *credentials*.

Cabe recalcar que el módulo *pandas* sirve para procesar conjuntos de datos usando objetos del tipo *DataFrame* que son tablas en las cuales se va a describir la información recopilada, y el módulo *json* sirve para tener una mejor representación de los objetos extraídos de Twitter y así poder acceder a cada uno de ellos.

Ahora, se procede a realizar la autenticación de la API usando las credenciales que fueron proporcionadas. Para ello, se usa las siguientes líneas de código:

```
# Autenticación
auth=tweepy.OAuthHandler(credentials.API_KEY, credentials.API_SECRET_
    KEY)
auth.set_access_token(credentials.ACCES_TOKEN,credentials.ACCES_TOKEN_
    SECRET)

api=tweepy.API(auth)
```

Figura 2.3: Autenticación para acceder a los servicios de la API.

Luego, se crea una función que va a contener los siguientes argumentos: *labe, petro, fecha_inicio,fecha_fin*. El argumento *labe* indica el nombre del ambiente de *Sandbox*, el argumento *petro* indica la frase con la cual se va a realizar la búsqueda en Twitter, los argumentos *fecha_inicio* y *fecha_fin* indican la fecha de inicio y la fecha final en las cuales se desea buscar respectivamente.

```
1  def buscar_tweets(labe,petro,fecha_inicio,fecha_fin):
2      i_json=[]
3
4      for i in tweepy.Cursor(api.search_full_archive ,label=labe,
5                             query=petro,fromDate=fecha_inicio,
6                             toDate=fecha_fin,maxResults=50).items(50):
7          i_json.append(json.loads(json.dumps(i._json)))
8
9      D_fecha={i.get('user').get('screen_name'):i.get('created_at') for i
       in i_json}
10     D_localizacion={i.get('user').get('screen_name'):i.get('user').get(
       'location') for i in i_json}
11     D_text={i.get('user').get('screen_name'):i.get('text') for i in i_
       json}
12
13
14     D1={'Fecha':D_fecha, 'Lugar':D_localizacion,'Texto':D_text}
15
16     return pd.DataFrame(D1)
```

Figura 2.4: Función para obtener los datos de la API.

Se puede ver que en las líneas 9, 10 y 11 de la función (2.4) se crean diccionarios, en los cuales se coloca la información acerca de la fecha, localización y el texto de cada tweet que la API vaya recolectando en la línea 4, luego se retorna una tabla de datos de tipo *DataFrame*[2] la cual tiene como columnas la fecha en la cual se realiza el reporte, el lugar desde el cual se realiza el reporte y el texto publicado en ese reporte.

Como la función tiene de argumentos la fecha de inicio y la fecha final para realizar las búsquedas, se crean varios intervalos de fechas en las cuales se va a realizar la búsqueda, en este componente se consideró que cada intervalo tenga de longitud 3 meses, esto debido a la limitación que presenta la API con respecto al número de peticiones que se le realice.

[2]Objeto que pertenece al módulo pandas que sirve para agrupar varios conjuntos de datos en una tabla.

```
# Intervalos de tiempo para recopilar datos

year=2018
mes=12
fechas=['201809280600']

while (year<2020 or mes<10):
    if mes>9:
        if mes<=12:
            fechas.append('{}{}280600'.format(year,mes))
            mes+=2
        if mes>12:
            mes-=12
            year+=1

    if mes<=9:
        fechas.append('{}0{}280600'.format(year,mes))
        mes+=2
```

Figura 2.5: Intervalos de tiempo para recopilar datos.

Notar que se debe usar la función *buscar_tweets* en cada intervalo de tiempo, por lo cual se tendrán varios *DataFrames*, por tanto, se consideró conveniente crear una lista con los *DataFrames* que retorne la función. Todo lo anterior se puede visualizar en las siguientes líneas de código.

```
# Extracción de datos usando la función creada

dftweets=[buscar_tweets('prod',petro='"derrame de petróleo en" "mar"',
          fecha_inicio=fechas[i],fecha_fin=fechas[i+1]) for i in range(
    len(fechas)) if i+1!=len(fechas)]
```

Figura 2.6: Lista de *DataFrames*.

El hecho de haber creado una lista de *DataFrames* permite usar la función *concat* que pertenece al módulo de *pandas* y sirve para concatenar todos los *DataFrames* en uno solo, y en conjunto con la función *sort_index()* se ordena el *DataFrame* con respecto al índice. Cabe recalcar que cada *DataFrame* tiene como índice al usuario que realizó el reporte.

```
# Concatenación de las datas

data_derrames=pd.concat(dftweets).sort_index()
```

Figura 2.7: Concatenación de los datos.

2.2. Preprocesamiento del conjunto de datos

Para procesar los datos ya recopilados se seguirán los siguientes pasos:

1. Obtener los países que tengan como frontera al Mar Caribe y al Golfo de México mediante técnicas de *web scraping*.
2. Obtener los sitios en los cuales se han reportado los derrames mediante el uso de expresiones regulares aplicadas a las observaciones de la variable **Texto** de la base de datos.
3. Determinar las coordenadas geográficas y los países de cada sitio antes encontrado.
4. Filtrar la base de datos obteniendo las observaciones correspondientes a los derrames de petróleo reportados en el Mar Caribe y el Golfo de México usando los países que se encontraron anteriormente.

2.2.1. Fronteras del Mar Caribe y el Golfo de México

El fin de conseguir las fronteras del Mar Caribe y el Golfo de México es poder filtrar todos los reportes conseguidos en la etapa de recolección de datos y solo tener los reportes de los derrames en nuestra zona de estudio.

Para obtener estas fronteras se usará la técnica de *web scraping* aplicada a las páginas web:

- **Caribe**: https://proyectoviajero.com/limites-del-caribe-fronteras/
- **Golfo de México**: https://www.ecured.cu/Golfo_de_M%C3%A9xico

Para realizar *web scraping* se usará los módulos *BeautifulSoup* y *requests*. *BeautifulSoup* es una librería de Python que permite extraer información de contenido en formato HTML o XML [7], es decir, tiene implementado el lenguaje XPath lo que permite acceder a cada uno de los elementos de una página web. Mientras que, la libería *requests* permite descargar el contenido de la página.

Para la instalación de ambas librerías se puede realizar mediante el comando *pip install beautifulsoup4* y *pip install requests*. Cabe recalcar que la librería *BeautifulSoup* tiene una dependencia con un módulo llamado *lxml*, el cual se encarga de interpretar XML, es por esto que también se debe instalar el módulo mediante el comando *pip install lxml*.

Se comienza importando las dos librerías:

```
from bs4 import BeautifulSoup
import requests as rq
```

Figura 2.8: Librerías *BeautifulSoup* y *requests*.

En el caso del Mar Caribe, la información de interés está en elementos del tipo lista que tiene como etiqueta *
...
* y dentro de la lista, cada elemento tiene la etiqueta *
...
*. Con esto podemos extraer la información necesaria mediante el siguiente código:

```
# obtener fronteras del Caribe
url='https://proyectoviajero.com/limites-del-caribe-fronteras/'
contenido_pg=rq.get(url)
soup_pg=BeautifulSoup(contenido_pg.content,'html.parser')
paises=soup_pg.find_all('ol')
paises1=[i.find_all('li') for i in paises]
paises2=[j.find_all('strong') for j in paises1[0]]
paises2=[i[0].text for i in paises2]
paises3=[z.text for z in paises1[1]]
paises_caribe=paises2+paises3[1:]
```

Figura 2.9: Código para obtener las fronteras del Mar Caribe.

En el caso del Golfo de México, la información de interés están en un elemento de tipo párrafo que tiene como etiqueta * ... * y dentro

del párrafo, los países se encuentran en elementos de tipo enlaces que tiene como etiqueta <a>.... Con esto podemos extraer la información necesaria usando el siguiente código:

```
#Obtener fronteras del Golfo de México
url='https://www.ecured.cu/Golfo_de_M%C3%A9xico'
contenido_pg=rq.get(url)
soup_pg=BeautifulSoup(contenido_pg.content,'html.parser')
pai_golfo=soup_pg.find_all('p')
pai_golfo1=pai_golfo[0].find_all('a')
paises_golfo=[i.text for i in pai_golfo1[0:3]]
```

Figura 2.10: Código para obtener las fronteras del Golfo de México.

2.2.2. Ubicación de los posibles derrames de petróleo

Si se analiza la información obtenida y que se encuentra organizada en la tabla creada como *DataFrame*, se puede notar que en la variable *Texto* los usuarios publican el lugar exacto en donde ocurrió un posible derrame. En la siguiente observación se puede visualizar el lugar entre todo el texto.

```
data_derrames.loc[1,'Texto']
✓ 0.9s                                                                    Python
'RT @AfpFactual: ¿Este video registra el reciente derrame de petróleo en el Parque Nacional de Morrocoy, en Venezuela? 🛢\n\n❌
No\n\nLas imágene…'
```

Figura 2.11: Ubicación del reporte de un posible derrame de petróleo.

Una manera sencilla de extraer el lugar del posible derrame de petróleo a partir del texto publicado es haciendo uso de expresiones regulares. Las expresiones regulares consisten en crear patrones de cadenas de caracteres y extraer todas las cadenas que cumplen con dichos patrones.

Python dispone de un módulo denominado *re*, el cual sirve para hacer uso de expresiones regulares. Hay que notar que en la variable *Texto* existe un patrón que viene dado de la forma:

([Dd]errame de petr[óo]leo en)\s(Av[.]\s[A-Z][\w\s.]|[\w\s.]*)*

donde los paréntesis sirven para agrupar en dos grupos, el primero para indicar que vaya la cadena *"derrame de petróleo"* con la condición de que la letra *d* puede estar en mayúscula o no, y de que la letra *o* puede estar tildada o no.

En el segundo grupo ya se obtiene el lugar de manera explicita, en el cual se usa el operador *or* de Python que se escribe por " | " el cual tomará cualquiera de los siguientes patrones que se cumpla: *Av[.]\s[A-Z][\w\s.]**, este patrón sirve para extraer cualquier avenida que se encuentre escrita en el texto; *[\w\s.]**, mientras que este patrón obtiene cualquier ubicación que no este antecedida por *"Av."*.

A continuación se presenta la implementación en el lenguaje Python.

```
# Expresiones regulares para obtener sitios
import re
import numpy as np
Texto=data_derrames['Texto']
sitios=[re.findall('([Dd]errame de petr[óo]leo en)\s(Av[.]\s[A-Z][\w\s
    .]*|[\w\s.]*)',i) for i in Texto]
lista_sitios=[i[0][1] if len(i)>0 else np.NAN for i in sitios]
data_derrames['Sitios']=lista_sitios
```

Figura 2.12: Uso de expresiones regulares para obtener sitios.

2.2.3. Ubicación geográfica de los posibles derrames de petróleo

Una vez ya extraídos los lugares en los cuales ocurrieron posibles derrames de petróleo, se procede a determinar sus coordenadas geográficas, pues estas serán necesarias para extraer las imágenes SAR de *Copernicus Acces Hub*.

Para obtener las coordenadas geográficas se hará uso de las librerías *basemap* y *geopy*, las cuales deben ser instaladas usando los comandos *pip install basemap* y *pip install geopy* respectivamente. Estas librerías permiten usar geolocalización para obtener la localización exacta de un lugar, ciudad o país como también sus coordenadas geográficas.

Con lo cual, se implementa el siguiente código que consta de la impor-

tación de la librería *geopy* junto con el uso de una API llamada *Nominatim*, la cual ofrece los servicios de geolocalización. También cabe recalcar que se hace uso de expresiones regulares, pues esto debido a que se encontró el siguiente patrón en los sitios encontrados anteriormente: *[A-Z][\w]** donde indica que puede ser cualquier cadena de caracteres que empiece con letra mayúscula y no haya espacios.

```
# Obtener coordenadas geográficas
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter
geolocator=Nominatim(user_agent='ubic')
geocode=RateLimiter(geolocator.geocode, min_delay_seconds=1)
data_derrames['Sitios']=[str(i) for i in data_derrames['Sitios']]
loc=[re.findall('[A-Z][\w]*',i) for i in data_derrames['Sitios']]
data_derrames['Sitios']=[' '.join(i) for i in loc]
data_derrames['location']=data_derrames['Sitios'].apply(geocode)
```

Figura 2.13: Geolocalización de los sitios en los cuales ocurrieron posibles derrames de petróleo.

Ahora, para determinar los países en los cuales hubo posibles derrames de petróleo se debe notar que en la nueva variable *location* del conjunto de datos, cada observación indica en un tipo de objeto *geopy.location.Location* toda la información acerca de los sitios usados en la geolocalización, es decir, también se incluye el país y las coordenadas geográficas.

La extracción de las coordenadas geográficas y de cada país se realiza mediante el siguiente código:

```
# Obtener países y coordenadas
df_data_derrames=data_derrames[data_derrames['location'].isnull()==
    False]
l=[tuple(df_data_derrames.loc[i,'location'])[1] for i in df_data_
    derrames.index]
l_pais=[tuple(df_data_derrames.loc[i,'location'])[0].split(' ')[-1] for
     i in df_data_derrames.index]
df_data_derrames['Coordenadas']=l
df_data_derrames['Pais']=l_pais
```

Figura 2.14: Coordenadas y países de los sitios en los cuales ocurrieron posibles derrames de petróleo.

2.2.4. Filtración de la base de datos

Una vez ya extraídos los países en los cuales ocurrieron posibles derrames de petróleo, se procede a filtrar solo la información de los países que se necesitan, esto es, los países que tienen como frontera al Mar Caribe y el Golfo de México.

Además se observar que algunas observaciones del conjunto de datos se repiten, esto es porque existieron varios usuarios reportando el mismo derrame de petróleo, por tanto, se debe filtrar para tener observaciones únicas.

Todo lo anterior se realiza mediante las siguientes líneas de código:

```
# Filtración de la Data
df_caribe=df_data_derrames[df_data_derrames.Pais.isin(paises)]
df_golfo=df_data_derrames[df_data_derrames.Pais.isin(paises_golfo)]
df_final=pd.concat([df_caribe, df_golfo]).sort_index()
```

Figura 2.15: Filtración de los países en la base de datos.

```
df_final.drop('location',axis=1, inplace=True)
df_final.drop_duplicates(inplace=True)
data_final_tic=df_final.drop(['Unnamed: 0.1','Lugar','Texto','Sitios'],
    axis=1)
data_final_tic['Latitud']=[i[0] for i in data_final_tic['Coordenadas']]
data_final_tic['Longitud']=[i[1] for i in data_final_tic['Coordenadas'
    ]]
data_final_tic.drop('Coordenadas',axis=1,inplace=True)
data_final_tic.drop_duplicates(subset=['Latitud','Longitud'],inplace=
    True)
data_final_tic.set_index('Pais',inplace=True)
```

Figura 2.16: Eliminación de observaciones duplicadas.

Notar que en la figura (2.16) también se presenta la eliminación de las variables *Texto*, *Lugar*, *Sitios*, *Unnamed: 0.1*. Cabe recalcar que la variable *Lugar* no fue usada ya que indica el lugar desde el cual se realiza el reporte, es decir, la ubicación del usuario, y la variable *Unnamed: 0.1* indica el nombre de usuario de cada persona que hizo el reporte.

En resumen se obtuvieron 5 observaciones debido a que la mayor parte del conjunto de datos tenía información que se repetía. A continuación se presenta como se realizó la extracción de las imágenes satelitales.

2.3. Extracción y procesamiento de imágenes satelitales

Para esta sección se necesitará hacer uso de las coordenadas geográficas encontradas en la sección anterior, esto debido a que la plataforma *Copernicus Open Acces Hub* dispone de un buscador de coordenadas y un filtro para fechas.

2.3.1. Extracción de imágenes SAR

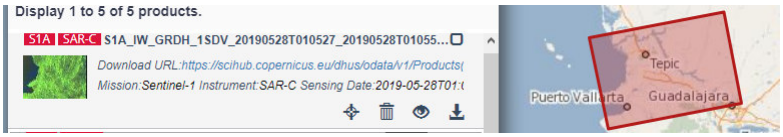
Para la extracción de las imágenes satelitales se usa la plataforma *Copernicus Open Acces Hub*, la cual dispone todas las misiones del Sentinel-1. Por tanto, se comienza accediendo al portal de *Copernicus* mediante

el enlace https://scihub.copernicus.eu/dhus/#/home y creando una cuenta en caso de no tenerla.

Una vez iniciada sesión, se realiza un polígono alrededor del punto que se tiene en la base de datos usando las opciones que da la plataforma, luego se da click en la opción de buscar obteniendo todas las misiones realizadas por los Sentineles alrededor del polígono graficado, luego se busca la fecha de interés y se descarga la imagen.

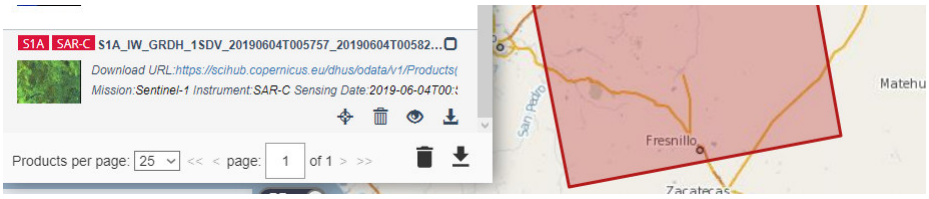
Para el desarrollo de este componente se descargó las 4 imágenes SAR que se indican en la siguiente figura.

(a) Cuba 11-10-2018.



(b) México 10-05-2019.

(c) Venezuela 27-08-2020.



(d) México 04-06-2019.

Figura 2.17: Recopilación de imágenes SAR en *Copernicus Acces Hub.*

En caso de que la imagen se encuentre en modo *Offline*, se deberá solicitar la imagen haciendo click en el boton de descarga y estar de acuerdo con el cuadro de diálogo, luego esperar hasta que aprueben la descarga de dicha imagen.

2.3.2. Procesamiento de imágenes SAR

Para el procesamiento de las imágenes SAR se puede usar la plataforma SNAP o el módulo *snappy*. La plataforma SNAP se la puede descargar desde el siguiente enlace: https://step.esa.int/main/download/snap-download/, esta plataforma permite procesar imágenes SAR de manera muy sencilla con el problema de no estar automatizado, es decir, un ser humano debe estar controlando el procesamiento de cada imagen.

En el desarrollo de este componente se uiliza el módulo *snappy* que se puede usar en Python, para su instalación se recomienda seguir la siguiente guía de usuario: https://rus-copernicus.eu/portal/wp-content/uploads/library/education/training/PY01_Sentinel1Processing_snappy.pdf

Se comienza importando las librerías y módulos necesarios,

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from zipfile import ZipFile
import os
from glob import iglob
import pandas as pd
import numpy as np
import snappy
import jpy
from termcolor import colored
```

Figura 2.18: Importación de librerías necesarias para el procesamiento de imágenes SAR.

Se debe crear una nueva carpeta, en este caso a la carpeta se le llamó “Imagen_sar” y en esta carpeta colocar todas las imágenes SAR descargadas anteriormente sin descomprimir. Luego se realiza la lectura de todas las imágenes mediante el siguiente código:

```
# Leer el producto

archivos=os.listdir('Imagen_sar')

name, sensing_mode, product_type, polarization, height, width, band_
    names=([] for i in range(7))

productos_leidos=[]

for i in archivos:
    sensing_mode.append(i.split('_')[3])
    product_type.append(i.split('_')[4])
    polarization.append(i.split('_')[-6])

    # snappy
    s1_read=snappy.ProductIO.readProduct('Imagen_sar/'+i)
    name.append(s1_read.getName())
    height.append(s1_read.getSceneRasterHeight())
    width.append(s1_read.getSceneRasterWidth())
    band_names.append(s1_read.getBandNames())

    productos_leidos.append(s1_read)
```

Figura 2.19: Lectura de imágenes SAR descargadas.

Luego, se toma un subconjunto de cada imagen para disminuir el tiempo de procesamiento y enfocarse solo en el área de interés.

```
# Subset
x,y,width,height=1200, 800, 15500, 15500

subset=[]
for i in range(5):
    parameters_subset=snappy.HashMap()
    parameters_subset.put('copyMetadata',True)
    parameters_subset.put('region','{},{},{},{}'.format(x,y,width,
    height))
    subset.append(snappy.GPF.createProduct('Subset',parameters_subset,
    productos_leidos[i]))
```

Figura 2.20: *subset* de cada imagen.

Para continuar con el procesamiento de cada imagen, se realiza la calibración radiométrica mediante las siguientes líneas de código.

```
# calibracion
calibrated=[]
for i in range(5):
    parameters_calibrate=snappy.HashMap()
    parameters_calibrate.put('outputSigmaBand',True)
    parameters_calibrate.put('sourceBands','Intensity_VH,Intensity_VV')
    parameters_calibrate.put('selectedPolarisations','VH,VV')
    parameters_calibrate.put('outputImageScaleInDb',False)
    calibrated.append(snappy.GPF.createProduct('Calibration',parameters
   _calibrate, subset[i]))
```

Figura 2.21: Calibración radiométrica de cada imagen.

Una vez realizada la calibración radiométrica se procede a aplicar el filtro *Lee Sigma* para reducir el ruido espectral y la textura habitual de cada imagen.

```
# Filtro Speckle
speckle=[]

for i in range(5):
    parametros=snappy.HashMap()
    parametros.put('filter','Lee Sigma')
    parametros.put('filterSizeX',5)
    parametros.put('filterSizeY',5)
    speckle.append(snappy.GPF.createProduct('Speckle-Filter',
   parametros, calibrated[i]))
```

Figura 2.22: Filtro espectral de cada imagen.

Finalmente, se realiza la exportación de las imágenes SAR en un formato *GeoTIF* el cual es un formato que toman las imágenes cuando se quiere realizar procesamiento geoespacial.

Figura 2.23: Exportación de imágenes procesadas.

Notar que las imágenes se están guardando en una carpeta llamada “imagenes_procesadas_nuevas”, con lo cual se debe tener creada esta carpeta en caso de usar el código.

En este componente no se realizó la detección de derrames de petróleo usando el algoritmo de detección implementado en *snappy* debido a problemas de rendimiento del equipo con el cual se realizó el componente, pero en el capítulo de anexos se adjunta el código en caso de que se requiera usar el algoritmo.

Capítulo 3

Resultados, conclusiones y recomendaciones

3.1. Resultados

En este capítulo se presentan los resultados principales que se obtuvieron en el capítulo 2, como por ejemplo: el conjunto de datos recopilados con la API de Twitter, los países que tienen como frontera al Mar Caribe y al Golfo de México, los lugares en los que ocurrieron posibles derrames de petróleo, ubicación geográfica de los lugares en los que reportaron posibles derrames.

Una vez que se presente todo lo relacionado a la base de datos con las respectivas coordenadas geográficas se procederá a presentar los resultados respecto a la extracción, procesamiento y clasificación de las imágenes satelitales.

3.1.1. Datos recopilados con la API de Twitter

Los datos que se obtuvieron al usar la API de Twitter son:

	Unnamed: 0	Fecha	Lugar	Texto
0	5berto	Mon Aug 24 17:50:22 +0000 2020	Caracas	RT @Kreico_: Científicos venezolanos lideran l...
1	71Emiliom	Thu Aug 27 17:38:04 +0000 2020	Vigo, España	RT @AfpFactual: ¿Este video registra el recien...
2	811gi	Fri Sep 27 21:24:29 +0000 2019	Houston, TX	RT @sumariumcom: Otro derrame de petróleo en l...
3	A1eGuerra	Fri Aug 02 04:50:45 +0000 2019	Santiago de Chile	RT @JaneGoodallCL: Derrame de petróleo en Chil...
4	AFPespanol	Thu Aug 27 14:16:51 +0000 2020	Montevideo, Uruguay	RT @AfpFactual: ¿Este video registra el recien...
...	...	...	...	...
325	wildaily	Tue Jul 30 21:17:24 +0000 2019	venezuela	RT @TuiteroSismico: Derrame de petróleo en #Ch...
326	yissi_2111	Sun Dec 01 13:43:26 +0000 2019	Santo Domingo D.N	RT @nuriapiera: Continúa derrame de petróleo e...
327	yoedmar	Fri Sep 27 18:11:48 +0000 2019	NaN	RT @sumariumcom: Otro derrame de petróleo en l...
328	zjefte	Thu Aug 27 14:18:28 +0000 2020	Oaxaca, México	RT @AfpFactual: ¿Este video registra el recien...
329	zuannyc	Mon Feb 24 05:20:12 +0000 2020	Bogotá, D.C., Colombia	RT @biommar: El mayor "derrame" de petróleo en...

330 rows × 4 columns

Figura 3.1: Conjunto de datos recopilados con Twitter.

En la cual se puede observar que existen 330 observaciones y las 4 variables mencionadas en la metodología.

3.1.2. Datos procesados

Para el procesamiento de los datos fue necesario identificar los sitios en los cuales se realizaron los reportes de posibles derrames de petróleo. Esto se lo realizó usando expresiones regulares en la variable *Texto* tal como se explicó en el anterior capítulo.

	Unnamed: 0	Fecha	Lugar	Texto	Sitios
0	5berto	Mon Aug 24 17:50:22 +0000 2020	Caracas	RT @Kreico_: Científicos venezolanos lideran l...	
1	71Emiliom	Thu Aug 27 17:38:04 +0000 2020	Vigo, España	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy
2	811gi	Fri Sep 27 21:24:29 +0000 2019	Houston, TX	RT @sumariumcom: Otro derrame de petróleo en l...	
3	A1eGuerra	Fri Aug 02 04:50:45 +0000 2019	Santiago de Chile	RT @JaneGoodallCL: Derrame de petróleo en Chil...	Chile
4	AFPespanol	Thu Aug 27 14:16:51 +0000 2020	Montevideo, Uruguay	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy

Figura 3.2: Sitios obtenidos mediante expresiones regulares.

Donde se puede ver que se ha creado una nueva variable denominada *Sitios*, la cual contiene a todos los lugares encontrados en la variable *Texto*.

Luego, se determinó los países que tienen como frontera al Mar Caribe y al Golfo de México mediante la técnica de *web scraping* obteniendo los siguientes resultados:

- **Mar Caribe**

```
['Antigua y Barbuda', 'Bahamas', 'Barbados', 'Cuba', 'Dominica', 'Granada', 'Haití', 'Jamaica', 'República Dominicana', 'San
Cristóbal y Nieves', 'San Vicente y las Granadinas', 'Santa Lucía', 'Trinidad y Tobago', 'Belice', 'Costa Rica', 'El Salvador',
'Guatemala', 'Honduras', 'Nicaragua', 'Panamá', 'Colombia', 'Venezuela', 'Guyana', 'Surinam', 'Guayana Francesa']
```

Figura 3.3: Países que tienen como frontera al Mar Caribe.

- **Golfo de México**

```
['México', 'Estados Unidos', 'Cuba']
```

Figura 3.4: Países que tienen como frontera al Golfo de México.

Luego, usando las observaciones de la variable *Sitios* en conjunto con las librerías de geolocalización *geopy* y *basemap* se obtienen las localizaciones con más exactitud, los resultados obtenidos son:

	Unnamed: 0.1	Fecha	Lugar	Texto	Sitios	location
1	71Emiliom	Thu Aug 27 17:38:04 +0000 2020	Vigo, España	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...
3	A1eGuerra	Fri Aug 02 04:50:45 +0000 2019	Santiago de Chile	RT @JaneGoodallCL: Derrame de petróleo en Chil...	Chile	(Chile, (-31.7613365, -71.3187697))
4	AFPespanol	Thu Aug 27 14:16:51 +0000 2020	Montevideo, Uruguay	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...
10	AfpFactual	Thu Aug 27 01:30:00 +0000 2020	Montevideo, Uruguay	¿Este video registra el reciente derrame de pe...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...
11	Alba2Fernandez	Mon Dec 02 09:32:57 +0000 2019	NaN	RT @nuriapiera: Continúa derrame de petróleo e...	Mar Caribe	(Caribbean Sea, (15.0000001, -75.0000001))
...	...	...	...	...	...	...
318	unfakingnews	Thu Aug 27 01:30:50 +0000 2020	Madrid, Spain	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...
320	valylewis	Mon Feb 03 18:23:28 +0000 2020	Gaiman Chubut, Argentina	RT @jlmiquelarena: El 26/12/2007 se provoco un...	Caleta Cordova	(Caleta Córdova, Municipio de Comodoro Rivadav...
323	wanly78	Mon Dec 02 18:45:16 +0000 2019	NaN	RT @nuriapiera: Continúa derrame de petróleo e...	Mar Caribe	(Caribbean Sea, (15.0000001, -75.0000001))
326	yissi_2111	Sun Dec 01 13:43:26 +0000 2019	Santo Domingo D.N	RT @nuriapiera: Continúa derrame de petróleo e...	Mar Caribe	(Caribbean Sea, (15.0000001, -75.0000001))
328	zjefte	Thu Aug 27 14:18:28 +0000 2020	Oaxaca, México	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...

136 rows × 6 columns

Figura 3.5: Localización de los sitios reportados de posibles derrames.

En la figura (3.5) se puede apreciar la existencia de una nueva variable denominada *location*, cada observación de esta variable presenta la ubicación completa de las observaciones de la variable *Sitios*. Además, el conjunto de datos presenta 136 observaciones de las 330 que eran inicialmente, esto debido a que no todos los sitios encontrados por expresiones regulares representaban a un lugar en específico, con lo cual se tuvieron que eliminar.

Con lo cual, se procede a separar en dos variables las coordenadas geográficas y el país en los cuales se reportaron posibles derrames de petróleo, de donde se obtiene lo siguiente:

	Unnamed: 0.1	Fecha	Lugar	Texto	Sitios	location	Coordenadas	Pais
1	71Emiliom	Thu Aug 27 17:38:04 +0000 2020	Vigo, España	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...	(10.850487000000001, -68.2491624481234)	Venezuela
3	A1eGuerra	Fri Aug 02 04:50:45 +0000 2019	Santiago de Chile	RT @JaneGoodallCL: Derrame de petróleo en Chil...	Chile	(Chile, (-31.7613365, -71.3187697))	(-31.7613365, -71.3187697)	Chile
4	AFPespanol	Thu Aug 27 14:16:51 +0000 2020	Montevideo, Uruguay	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...	(10.850487000000001, -68.2491624481234)	Venezuela
10	AfpFactual	Thu Aug 27 01:30:00 +0000 2020	Montevideo, Uruguay	¿Este video registra el reciente derrame de pe...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...	(10.850487000000001, -68.2491624481234)	Venezuela
11	Alba2Fernandez	Mon Dec 02 09:32:57 +0000 2019	NaN	RT @nuriapiera: Continúa derrame de petróleo e...	Mar Caribe	(Caribbean Sea, (15.0000001, -75.0000001))	(15.0000001, -75.0000001)	Sea
...	...	...	...	...	...	...	...	...
318	unfakingnews	Thu Aug 27 01:30:50 +0000 2020	Madrid, Spain	RT @AfpFactual: ¿Este video registra el recien...	Parque Nacional Morrocoy	(Parque Nacional Morrocoy, Municipio Silva, Fa...	(10.850487000000001, -68.2491624481234)	Venezuela

Figura 3.6: Coordenadas y país donde ocurrieron los posibles derrame.

Y filtrando la data con respecto a los países que tienen como frontera al Mar Caribe y al Golfo de México, además eliminando los datos duplicados se tiene la siguiente data final:

Pais	Fecha	Latitud	Longitud
Venezuela	Thu Aug 27 17:38:04 +0000 2020	10.850487	-68.249162
México	Fri May 10 01:49:43 +0000 2019	20.748945	-105.250729
Cuba	Thu Oct 11 17:20:38 +0000 2018	22.185642	-81.169388
México	Tue Jun 04 03:04:36 +0000 2019	23.658512	-102.007710

Figura 3.7: Conjunto de datos final

En la cual se puede observar que existen solo 4 observaciones, esto

debido a la alta cantidad de usuarios que realizan el reporte de un mismo derrame provocando que haya datos duplicados. Además se eliminó las variables no necesarias.

En el siguiente diagrama de barras se puede visualizar la cantidad de datos duplicados que se tenía en la data, donde se tiene que el $87{,}88\,\%$ de los datos son duplicados.

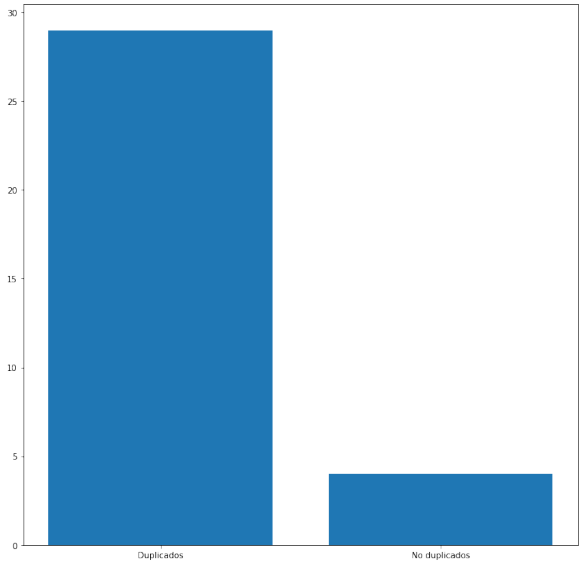


Figura 3.8: Datos duplicados y únicos.

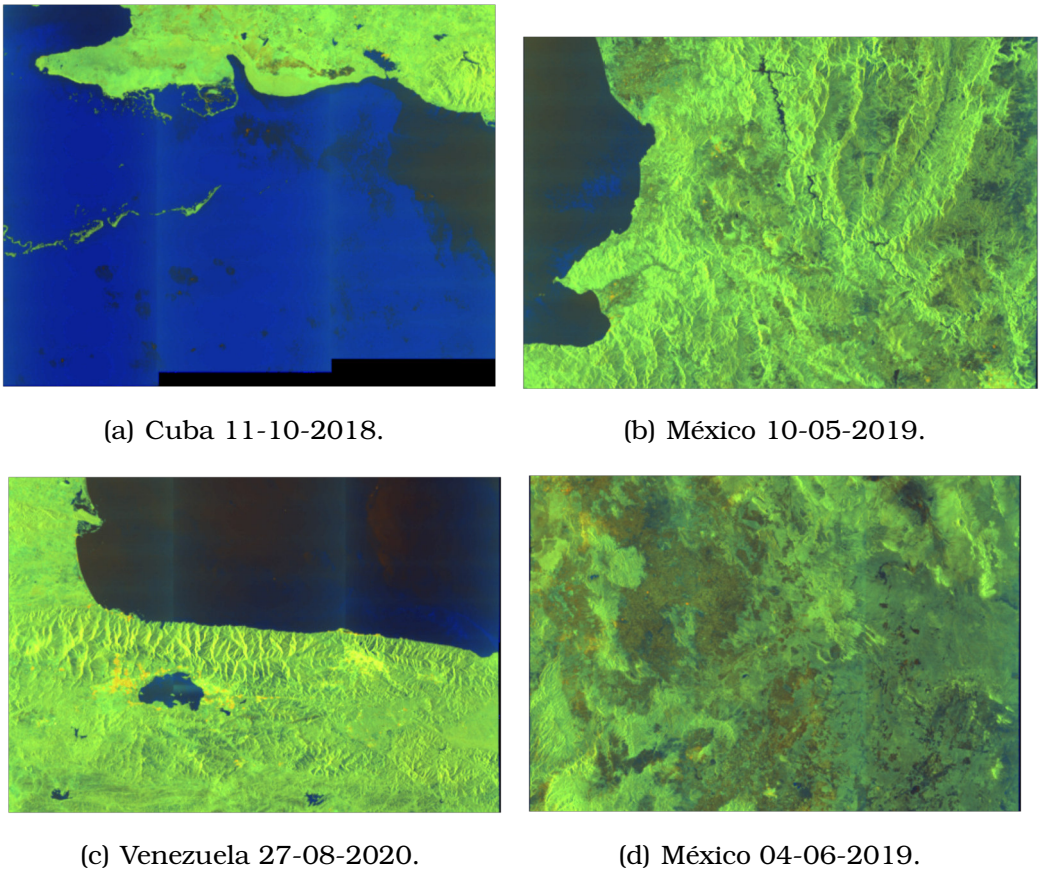
Ahora, representando las coordenadas geográficas en un mapa del continente de América se tiene que:

Figura 3.9: Coordenadas representadas en el mapa de América.

donde se puede ver que una observación del país de México del 4 de junio del 2019 no corresponde a un reporte de derrame en el Mar. Mientras que la otra observación del país de México corresponde al mar pero no al Mar Caribe o al Golfo de México.

3.1.3. Imágenes satelitales recopiladas

Las imágenes satelitales recopiladas de los cuatro lugares de la base de datos son:



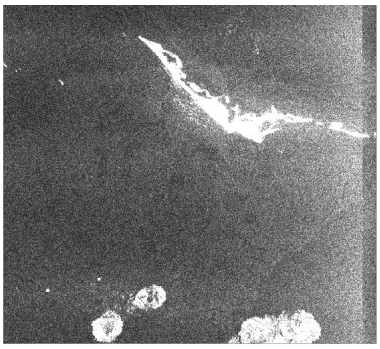
(a) Cuba 11-10-2018. (b) México 10-05-2019.

(c) Venezuela 27-08-2020. (d) México 04-06-2019.

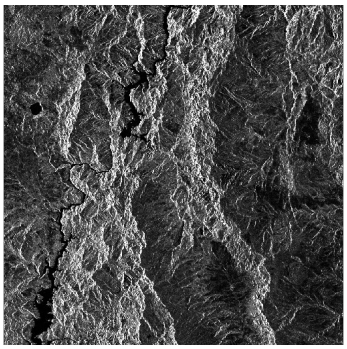
Figura 3.10: Imágenes satelitales extraídas de *Copernicus Open Acces Hub.*

3.1.4. Imágenes satelitales procesadas

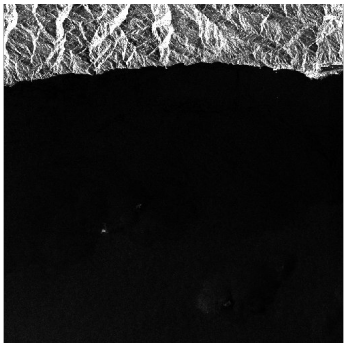
Las imágenes obtenidas al procesarlas aplicando una calibración y el filtro *speckle* son:



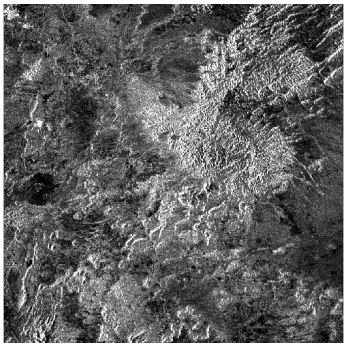
(a) Cuba 11-10-2018.



(b) México 10-05-2019.



(c) Venezuela 27-08-2020.



(d) México 04-06-2019.

Figura 3.11: Imágenes satelitales procesadas.

3.2. Conclusiones y recomendaciones

3.2.1. Conclusiones

Tras desarrollar el trabajo de integración curricular, se tiene claro que el uso de las nuevas tecnologías como lo son las APIS pueden ayudar a realizar estudios en Ciencia de Datos, pues la API de Twitter permitió obtener de manera más rápida los reportes de posibles derrames de petróleo que hayan ocurrido.

Además, cabe notar que en la red social de Twitter, un suceso puede ser publicado por una gran cantidad de personas, esto se refleja en que la mayor parte de los datos recopilados, pues estos fueron datos repetidos, lo cual se puede ver en el capítulo de resultados en la figura (3.8) donde se indica que existen un $87{,}88\,\%$ de datos repetidos.

El uso de la técnica de *web scraping* permite obtener cualquier tipo de

información en una página web con tan solo de conocer su estructura, tal como se realizó en este componente, permitió obtener todos los países que tienen como frontera al Mar Caribe y al Golfo de México permitiendo filtrar la base de datos a nuestra área de interés.

Obtener las coordenadas geográficas de los lugares en los cuales se reportaron posibles derrames de petróleo fue un aspecto muy importante en el desarrollo de este componente, pues con esto se pudo realizar la recopilación de las imágenes satelitales en la plataforma *Copernicus Open Acces Hub*, las cuales se encontraban en modo *offline*, lo cual indica que se debe solicitar la imagen y esperar a que aprueben esta solicitud, esto afectó en el tiempo de desarrollo del componente.

El módulo *snappy* permitió procesar las imágenes satelitales de manera más rápida que la plataforma SNAP, esto debido a que usamos el lenguaje de programación Python, pues se logró crear un programa que puede procesar varias imágenes SAR con tan solo ejecutar el programa, mientras que en la plataforma SNAP toca realizar todos los pasos de procesamiento de manera manual.

Los resultados obtenidos en el procesamiento de las imágenes fueron los esperados al filtro *Lee Sigma* para eliminar el ruido espectral de las imágenes SAR, y se obtiene una mayor visibilidad de en escala de grises.

Se logró crear una base de datos que consta de 4 observaciones, cada una especificando el país en los cuales ocurrieron los derrames de petróleo, la fecha, la ubicación geográfica y sus correspondientes imágenes SAR.

3.2.2. Recomendaciones

Con respecto a la recopilación de datos usando la API de Twitter se recomienda realizar las búsquedas en un intervalo de tiempo más amplio y los periodos más pequeños, para así conseguir mayor información útil.

Al momento de procesar la información recopilada de Twitter se recomienda analizar un poco más los patrones que existen en la variable *Texto* para evitar perder información importante, y así conseguir un mejor filtrado de la base de datos.

El procesamiento de imágenes satelitales consume mucho tiempo y demasiados recursos computacionales, por lo que se recomienda para estudios posteriores definir subconjuntos de la imagen más pequeños y realizarlo en equipo.

Debido al problema que se tiene para procesar las imágenes SAR, en estudios posteriores se recomienda considerar crear un modelo de clasificación analizando el comportamiento de los usuarios de Twitter al publicar un suceso verdadero y un suceso falso.

Capítulo A

Anexos

En este capitulo de Anexos se adjunta los códigos completos con los cuales se desarrolló el trabajo de integración curricular.

A.1. Recopilación de datos en *Twitter*

```
import tweepy
import credentials
import pandas as pd
import json

# Autenticación
auth=tweepy.OAuthHandler(credentials.API_KEY, credentials.API_SECRET_
    KEY)
auth.set_access_token(credentials.ACCES_TOKEN,credentials.ACCES_TOKEN_
    SECRET)

api=tweepy.API(auth)

# Función para extraer datos

def buscar_tweets(labe,petro,fecha_inicio,fecha_fin):
    i_json=[]

    for i in tweepy.Cursor(api.search_full_archive ,label=labe,
                           query=petro,fromDate=fecha_inicio,
```

```
                                toDate=fecha_fin,maxResults=50).items(50):
        i_json.append(json.loads(json.dumps(i._json)))

    D_fecha={i.get('user').get('screen_name'):i.get('created_at') for i
     in i_json}
    D_localizacion={i.get('user').get('screen_name'):i.get('user').get(
    'location') for i in i_json}
    D_text={i.get('user').get('screen_name'):i.get('text') for i in i_
    json}


    D1={'Fecha':D_fecha, 'Lugar':D_localizacion,'Texto':D_text}

    return pd.DataFrame(D1)

# Intervalos de tiempo para recopilar datos

year=2018
mes=12
fechas=['201809280600']

while (year<2020 or mes<10):
    if mes>9:
        if mes<=12:
            fechas.append('{}{}280600'.format(year,mes))
            mes+=2
        if mes>12:
            mes-=12
            year+=1

    if mes<=9:
        fechas.append('{}0{}280600'.format(year,mes))
        mes+=2

# Extracción de datos usando la función creada

dftweets=[buscar_tweets('prod',petro='"derrame de petróleo en" "mar"',
          fecha_inicio=fechas[i],fecha_fin=fechas[i+1]) for i in range(
    len(fechas)) if i+1!=len(fechas)]

# Concatenación de las datas

data_derrames=pd.concat(dftweets).sort_index()
```

A.2. Procesamiento de datos obtenidos en *Twitter*

```
from bs4 import BeautifulSoup
import requests as rq


# obtener fronteras del Caribe
url='https://proyectoviajero.com/limites-del-caribe-fronteras/'
contenido_pg=rq.get(url)
soup_pg=BeautifulSoup(contenido_pg.content,'html.parser')
paises=soup_pg.find_all('ol')
paises1=[i.find_all('li') for i in paises]
paises2=[j.find_all('strong') for j in paises1[0]]
paises2=[i[0].text for i in paises2]
paises3=[z.text for z in paises1[1]]
paises_caribe=paises2+paises3[1:]


#Obtener fronteras del Golfo de México
url='https://www.ecured.cu/Golfo_de_M%C3%A9xico'
contenido_pg=rq.get(url)
soup_pg=BeautifulSoup(contenido_pg.content,'html.parser')
pai_golfo=soup_pg.find_all('p')
pai_golfo1=pai_golfo[0].find_all('a')
paises_golfo=[i.text for i in pai_golfo1[0:3]]

# Expresiones regulares para obtener sitios
import re
import numpy as np
Texto=data_derrames['Texto']
sitios=[re.findall('([Dd]errame de petr[óo]leo en)\s(Av[.]\s[A-Z][\w\s
    .]*|[\w\s.]*)',i) for i in Texto]
lista_sitios=[i[0][1] if len(i)>0 else np.NAN for i in sitios]
data_derrames['Sitios']=lista_sitios

# Obtener coordenadas geográficas
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter
geolocator=Nominatim(user_agent='ubic')
geocode=RateLimiter(geolocator.geocode, min_delay_seconds=1)
```

```
data_derrames['Sitios']=[str(i) for i in data_derrames['Sitios']]
loc=[re.findall('[A-Z][\w]*',i) for i in data_derrames['Sitios']]
data_derrames['Sitios']=[' '.join(i) for i in loc]
data_derrames['location']=data_derrames['Sitios'].apply(geocode)

# Obtener países y coordenadas
df_data_derrames=data_derrames[data_derrames['location'].isnull()==
    False]
l=[tuple(df_data_derrames.loc[i,'location'])[1] for i in df_data_
    derrames.index]
l_pais=[tuple(df_data_derrames.loc[i,'location'])[0].split(' ')[-1] for
     i in df_data_derrames.index]
df_data_derrames['Coordenadas']=l
df_data_derrames['Pais']=l_pais

# Filtración de la Data
df_caribe=df_data_derrames[df_data_derrames.Pais.isin(paises)]
df_golfo=df_data_derrames[df_data_derrames.Pais.isin(paises_golfo)]
df_final=pd.concat([df_caribe, df_golfo]).sort_index()
df_final.drop('location',axis=1, inplace=True)
df_final.drop_duplicates(inplace=True)
data_final_tic=df_final.drop(['Unnamed: 0.1','Lugar','Texto','Sitios'],
     axis=1)
data_final_tic['Latitud']=[i[0] for i in data_final_tic['Coordenadas']]
data_final_tic['Longitud']=[i[1] for i in data_final_tic['Coordenadas'
    ]]
data_final_tic.drop('Coordenadas',axis=1,inplace=True)
data_final_tic.drop_duplicates(subset=['Latitud','Longitud'],inplace=
    True)
data_final_tic.set_index('Pais',inplace=True)
```

A.3. Procesamiento de imágenes SAR

```
from glob import iglob
from itertools import product
import matplotlib.colors as colors
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from zipfile import ZipFile
import os
from glob import iglob
import pandas as pd
```

```
import numpy as np
import snappy
import jpy
from termcolor import colored

# funcion para grafica
def output_view(product,band,min_value_VV,max_value_VV,min_value_VH,max_value_VH):
    band_data_list=[]

    for i in band:
        band=product.getBand(i)
        w=band.getRasterWidth()
        h=band.getRasterHeight()
        band_data=np.zeros(w*h,np.float)
        band.readPixels(0,0,w,h,band_data)
        band_data.shape=h,w
        band_data_list.append(band_data)

    fig, (ax1,ax2)=plt.subplots(1,2,figsize=(16,16))
    ax1.imshow(band_data_list[0], cmap='gray' , vmin=min_value_VV, vmax=max_value_VV)
    ax1.set_title(output_bands[0])
    ax2.imshow(band_data_list[1], cmap='gray', vmin=min_value_VH, vmax=max_value_VH)
    ax2.set_title(output_bands[1])

    for ax in fig.get_axes():
        ax.label_outer()
# Leer el producto

archivos=os.listdir('Imagen_sar')

name, sensing_mode, product_type, polarization, height, width, band_names=([] for i in range(7))

productos_leidos=[]

for i in archivos:
    sensing_mode.append(i.split('_')[3])
    product_type.append(i.split('_')[4])
    polarization.append(i.split('_')[-6])

```

```
    # snappy
    s1_read=snappy.ProductIO.readProduct('Imagen_sar/'+i)
    name.append(s1_read.getName())
    height.append(s1_read.getSceneRasterHeight())
    width.append(s1_read.getSceneRasterWidth())
    band_names.append(s1_read.getBandNames())

    productos_leidos.append(s1_read)

data_s1_read=pd.DataFrame({'Nombre':name,'Sensing Mode':sensing_mode,'Product Type':product_type,
                           'Polarization':polarization, 'Height':height,'Width':width,
                           'Band Names':band_names})

print(data_s1_read)

data_s1_read.to_excel('datas_s1/data_1.xlsx')

with ZipFile('Imagen_sar/'+archivos[0],'r') as qck_look:
    qck_look=qck_look.open(name[0]+'.SAFE/preview/quick-look.png')
    img=mpimg.imread(qck_look)
    plt.figure(figsize=(15,15))
    plt.title('Visualizacion - '+name[0]+'\n')
    plt.axis('off')
    plt.imshow(img)

# Subset
x,y,width,height=1200, 800, 15500, 15500

subset=[]
for i in range(5):
    parameters_subset=snappy.HashMap()
    parameters_subset.put('copyMetadata',True)
    parameters_subset.put('region','{},{},{},{}'.format(x,y,width,height))
    subset.append(snappy.GPF.createProduct('Subset',parameters_subset,productos_leidos[i]))

# calibracion
calibrated=[]
for i in range(5):
    parameters_calibrate=snappy.HashMap()
```

```
    parameters_calibrate.put('outputSigmaBand',True)
    parameters_calibrate.put('sourceBands','Intensity_VH,Intensity_VV')
    parameters_calibrate.put('selectedPolarisations','VH,VV')
    parameters_calibrate.put('outputImageScaleInDb',False)
    calibrated.append(snappy.GPF.createProduct('Calibration',parameters
    _calibrate, subset[i]))


# Filtro Speckle
speckle=[]

for i in range(5):
    parametros=snappy.HashMap()
    parametros.put('filter','Lee Sigma')
    parametros.put('filterSizeX',5)
    parametros.put('filterSizeY',5)
    speckle.append(snappy.GPF.createProduct('Speckle-Filter',
    parametros, calibrated[i]))


# Exportacion
for i in range(5):
    snappy.ProductIO.writeProduct(speckle[i],'imagenes_procesadas_
    nuevas/imagen_procesada{}'.format(i),'GeoTIFF')
    print(colored('Producto guardado exitosamente en:','green'),'
    imagenes_procesadas_nuevas/imagen_procesada{}'.format(i))
```

Referencias bibliográficas

[1] B Bhandari. Processing sentinel-1 sar images using sentinel application platform (snap). *Geomatics, remote*, 2019.

[2] G Calabresi, F Del Frate, J Lichtenegger, A Petrocchi, and P Trivero. Neural networks for the oil spill detection using ers-sar data. In *IEEE 1999 International Geoscience and Remote Sensing Symposium. IGARSS'99 (Cat. No. 99CH36293)*, volume 1, pages 215–217. IEEE, 1999.

[3] Alvaro Chirino Gutierrez. *Introducción al Big Data con R*. 2021.

[4] Iphigenia Keramitsoglou, Constantinos Cartalis, and Chris T Kiranoudis. Automatic identification of oil spills on satellite images. *Environmental modelling & software*, 21(5):640–652, 2006.

[5] Martin Kirscht and Carsten Rinke. 3d reconstruction of buildings and vegetation from synthetic aperture radar (sar) images. In *MVA*, pages 228–231. Citeseer, 1998.

[6] Jaime López. Web scraping. 2018.

[7] Juan José Lozano. Web scraping con python. guía de inicio de beautiful soup, Jan 2022.

[8] Mostafa Mansourpour, MA Rajabi, and JAR Blais. Effects and performance of speckle noise reduction filters on active radar and sar images. In *Proc. Isprs*, volume 36, page W41, 2006.

[9] Barbara Ornitz and Michael Champ. *Oil spills first principles: prevention and best response*. Elsevier, 2002.

[10] Adrián Peña-Peñate, Luis Guillermo Silva Rojas, and Rubén Alcolea Núñez. Módulo de filtrado y segmentación de imágenes médicas digitales para el proyecto vismedic. *Revista Cubana de Ciencias Informáticas*, 10(1):13–27, 2016.

[11] SIGMUR. *Teledetección*. Universidad de Murcia, 2003.

[12] Dan Wang and Murtaza Ali. Synthetic aperture radar on low power multi-core digital signal processor. In *2012 IEEE Conference on High Performance Extreme Computing*, pages 1–6. IEEE, 2012.