

Proyecto Final. Análisis de Sargazo

(Diciembre, 2021)

1st Guerra Silva Erick Iván
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
erickivanguerra0.0@gmail.com

3rd Martínez Gutiérrez Carlos Giovanni
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
cgiovanni@comunidad.unam.mx

2nd Lázaro Martínez Abraham Josué
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
abrahamlazaro@comunidad.unam.mx

4th Castillo Sánchez Axel
Ingeniería Mecatrónica
Universidad Nacional Autónoma de México
 Ciudad de México, México
axel_castillo98@hotmail.com

Resumen—Este documento presenta los resultados obtenidos en la realización del proyecto final, el cual tenía como objetivo el análisis de imágenes de sargazo. Todo el proyecto se realizó en lenguaje Python utilizando Google Collaboratory y se utilizaron módulos como MultinomialNB de Sklearn para la creación de un clasificador de Bayes, Neighbors también de Sklearn para la implementación de un clasificador K-NN, al igual que el módulo cross_val_score de Sklearn para medir el desempeño de los resultados obtenidos.

Índice de Términos—Procesamiento de Imágenes, caracterización y clasificación, clasificador, texturas, validación, distancia, ángulo, Haralick, segmentación, matriz, sargazo.

I. INTRODUCCIÓN

Sargazo.

El sargazo (*Sargassum*) es una especie de alga parda que habita en ambientes marinos de las zonas tropicales del mundo.

Se caracteriza por poseer unas estructuras llamadas aerocistos, pequeñas bolas llenas de gases que ayudan al sargazo a flotar y no hundirse, es por eso que se les ve en grandes cantidades flotando en las playas.

Los aerocistos transforman el CO₂ y el agua en azúcares, las cuales utilizan el sargazo como fuente de energía. [1]

Análisis y caracterización de texturas

El análisis de texturas hace referencia a la caracterización de las regiones de una imagen por su contenido de textura. El análisis de texturas intenta cuantificar las cualidades intuitivas descritas por términos como áspero, suave, sedoso o accidentado en función de la variación espacial en las intensidades de píxeles. En este sentido, la rugosidad o bache se refiere a variaciones en los valores de intensidad, o niveles de gris.

El análisis de texturas puede ser útil cuando los objetos de una imagen se caracterizan más por su textura que por la intensidad, y las técnicas de umbral tradicionales no se pueden utilizar de forma eficaz.[2]

Métodos de validación

La validación de un método es el proceso para confirmar que el procedimiento analítico utilizado para una prueba en concreto es adecuado para su uso previsto. Los resultados de la validación del método pueden utilizarse para juzgar la calidad, la fiabilidad y la constancia de los resultados analíticos, se trata de una parte integrante de cualquier buena práctica analítica.[3]

Clasificadores K-NN

El K-NN es un algoritmo de aprendizaje supervisado,

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

es decir, que a partir de un juego de datos inicial su objetivo será el de clasificar correctamente todas las instancias nuevas. El juego de datos típico de este tipo de algoritmos está formado por varios atributos descriptivos y un solo atributo objetivo (también llamado clase).

K-NN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se prueban los datos de test.[4]

K-Means

K-means es un algoritmo de clasificación no supervisada (clusterización) que agrupa objetos en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster. Se suele usar la distancia cuadrática.[5]

El algoritmo consta de tres pasos:

- Inicialización: una vez escogido el número de grupos, k, se establecen k centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente.
- Asignación objetos a los centroides: cada objeto de los datos es asignado a su centroide más cercano.
- Actualización centroides: se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.

Regresión logística simple y multiple.

La Regresión Logística Simple, desarrollada por David Cox en 1958, es un método de regresión que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa. Una de las principales aplicaciones de la regresión logística es la de clasificación binaria, en el que las observaciones se clasifican en un grupo u otro dependiendo del valor que tome la variable empleada como predictor.

La regresión logística múltiple es una extensión de la regresión logística simple. Se basa en los mismos principios que la regresión logística simple pero ampliando el número de predictores. Los predictores pueden ser tanto continuos como categóricos. [6]

Rede neuronales

Una red neuronal es un modelo simplificado que emula el modo en que el cerebro humano procesa la información: Funciona simultaneando un número elevado de unidades de procesamiento interconectadas que parecen versiones abstractas de neuronas.

Las unidades de procesamiento se organizan en capas. Hay tres partes normalmente en una red neuronal : una capa de entrada, con unidades que representan los campos de entrada; una o varias capas ocultas; y una capa de salida, con una unidad o unidades que representa el campo o los campos de destino. Las unidades se conectan con fuerzas de conexión variables (o ponderaciones). Los datos de entrada se presentan en la primera capa, y los valores se propagan desde cada neurona hasta cada neurona de la siguiente capa. al final, se

envía un resultado desde la capa de salida. [7]

Optimizador Adam

El descenso de gradiente es un algoritmo de optimización que sigue el gradiente negativo de una función objetivo para localizar el mínimo de la función.

Una limitación del descenso de gradiente es que se usa un tamaño de paso único (tasa de aprendizaje) para todas las variables de entrada. Las extensiones al descenso de gradiente como AdaGrad y RMSProp actualizan el algoritmo para usar un tamaño de paso separado para cada variable de entrada, pero pueden dar como resultado un tamaño de paso que disminuye rápidamente a valores muy pequeños.

La Estimación de movimiento adaptativo algoritmo, o Adam para abreviar, es una extensión del descenso de gradiente y un sucesor natural de técnicas como AdaGrad y RMSProp que adapta automáticamente una tasa de aprendizaje para cada variable de entrada para la función objetivo y suaviza aún más el proceso de búsqueda mediante el uso de una media móvil exponencialmente decreciente del gradiente para realizar actualizaciones a las variables. [8]

Entropía cruzada

La entropía cruzada es una métrica que puede utilizarse para reflejar la precisión de los pronósticos probabilísticos y está estrechamente vinculada con la estimación por máxima verosimilitud. La entropía cruzada es de gran importancia para los sistemas pronóstico modernos, porque es esencial para la entrega de pronósticos superiores, incluso para métricas alternativas. Desde una perspectiva de cadena de suministro, la entropía cruzada es particularmente importante, porque respalda el cálculo de modelos que también son buenos para la captura de posibilidades de eventos raros, que a menudo resultan ser los más costosos. Esta métrica se aleja bastante de la idea que defiende métricas de precisión más simples, como el error cuadrático medio o el porcentaje de error absoluto medio. [9]

II. OBJETIVOS

El alumno:

- Clasificación de la escena incluyendo al sargazo como una de las clases o Segmentación de la(s) mancha(s) de sargazo.
- Cuantificación (en píxeles) del sargazo.

III. RESULTADOS

A continuación, se presentan los ejercicios desarrollados en este proyecto.

Parte A SEGMENTACIÓN (Utilización de clasificadores)

Para poder realizar la segmentación de las imágenes proporcionadas por la profesora, utilizaremos la matriz GLCM, y los parámetros que se utilizaran, serán los siguientes.

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

```
# distancia de comparación entre pixeles
distancia = 8
# tamaño de la ventana
n = 32
# angulo de comparaciones
angulo = [0, 45, 90]
# blur
blur = 31
bg_x = 51
bg_y = 51
bg_s = 0
escala = 256
```

La variable distancia, representa la distancia en que se realizará cada ventana, la variable n, representa el tamaño de la ventana, la variable ángulo, representa la forma en que se realizará la matriz, sin embargo la función que realiza la matriz, puede aceptar más de un ángulo para la creación de la misma, posteriormente se encuentran las variables para poder aplicar el blur y por último la variable escala, la cual cuenta con el número total de escala de grises.

Para la generación de la matriz de Haralick a partir de las ventanas de cada imagen se utilizaron las siguientes funciones.

GLCM

Esta función nos ayudará a crear la matriz de Haralick, recibe 4 parámetros, los cuales son m (matriz inicial), distancia (distancia de movimiento entre los pixeles de la imagen), angulo (recibe los ángulos con los que se elaborará la matriz), escala (Tamaño de la matriz de co-ocurrencia).

```
def GLCM(matriz,distancia,angulo,escala):
```

Después de que la función obtiene los cuatro parámetros, se crea la matriz glcm, llenándola con ceros, además de crear la matriz inicial, a partir del ancho y el alto de la imagen a analizar.

Posteriormente se comienza a recorrer la imagen para comenzar a llenar la matriz glcm, esto se llenará dependiendo del ángulo que se escogió, sin embargo, esta función acepta más de dos ángulos para poder trabajar, a continuación se muestra el llenado de la matriz glcm, cuando el ángulo es cero.

```
for x in range(width):
    for y in range(height):
        if A==0:#Dirección 0 °
            if y+distancia < height:
                glcm[m[x][y]][m[x][y+distancia]]+=1
```

Al terminar este proceso, la función regresará la matriz glcm, sin embargo en el final se utilizará otra función para poder obtener los estadísticos de segundo orden.

Para obtener los estadísticos de segundo orden de la matriz glcm, se utiliza la función vectorizar.

```
def vectorizar(glcm):
    var = glcm.var()
    mean = glcm.mean()
    std = glcm.std()
    glcm_h = glcm.reshape((-1,1))
    h = entropy(glcm_h, base=2)
    return [mean, var, std, h[0]]
```

Esta función obtendrá los estadísticos de segundo orden de cada ventana a partir de la matriz glcm, los estadísticos que se obtendrán, serán la media, varianza, desviación estándar y la entropía.

La siguiente función utilizada es la función generar pruebas, esta función nos ayudará a crear las ventanas en cada imagen, los parámetros que recibe esta función son siete, la imagen, la distancia para crear las ventanas, el ángulo para generar matriz, la escala, el tamaño de la imagen (X Y) y la variable n, la cual sera el tamaño de la ventana.

```
def generar_pruebas(imagen, distancia, angulo,
escala, tamaño_x, tamaño_y=None, n=None):
```

Al inicio de la función, tendremos un if, en donde se validará si la variable n contiene algo, si esta variable, no contiene un dato, se realizará la siguiente operación para obtenerla.

```
n = distancia*2+1
```

La siguiente validación que se realizará, será el contenido de la variable tamaño_y, si esta variable es none, se le asignará el valor de tamaño_x.

Por lo tanto si se manda una distancia de 2, la ventana que obtendremos será de 5x5, para poder obtener las ventanas, se va recorriendo la matriz de la imagen, a partir del tamaño ingresado, posteriormente, cada ventana se guarda en una lista generada al inicio de la función.

```
for i in range(0,tam_x-n,distancia):
    for j in range(0,tam_y-n,distancia):
        matriz = imagen[i:i+n,j:j+n,:]
```

Después de ir recorriendo la matriz de la imagen a partir de las ventanas e ir generando la matriz de la ventana, se validara que la suma de esta ventana sea mayor a 100, esta validacion se hace, para poder ocrroborar la cantidad de pixeles negros que tiene esa venta, si la ventana tiene una suma mayor a 100, se aplicara la funcion GLCM, en este caso se realizara esta matriz para los tres diferentes canales de nuestra imagen, estas matrices resultantes se sumaran para obtener una sola matriz y al final, la función regresará las ventanas en una lista.

```
R_GLCM = GLCM(matriz[:, :, 0], distancia, angulo,
escala)
G_GLCM = GLCM(matriz[:, :, 1], distancia, angulo,
escala)
B_GLCM = GLCM(matriz[:, :, 2], distancia, angulo,
escala)
Vector_Prediccion
```

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

```
=np.array([R_GLCM+G_GLCM+B_GLCM])
```

También se utiliza la función etiquetar, esta función nos ayuda a etiquetar las ventanas con la imagen de entrenamiento a las que corresponden y no mezclar las ventanas de cada clase o imagen asociada.

Todas las funciones descritas anteriormente, se utilizan en una función que hace todo lo descrito anteriormente, esta función se llama sacar pruebas, esta función recibe cuatro parámetros, la url de la imagen, la distancia para crear las ventanas, el ángulo o ángulos para obtener la matriz glcm y la etiqueta o clase asociada.

Primero se lee la imagen a analizar, a partir de la url que recibe la función.

```
img = cv2.imread(url)
```

Posteriormente se cambiará el formato de la imagen de BGR a RGB y se utilizará la función aplicar blur, esta función recibe cinco parámetros, el primero es la imagen, el segundo y tercer parámetro será el tamaño del filtro, por último el cuarto parámetro será la desviación estándar del filtro, dentro de la función se validará que el tamaño del filtro no sea un número par y al finalizar se retorna la imagen con el filtro ya aplicado.

```
def aplicar_blur(img, gb_x, gb_y=None, gb_s=0):
    if gb_x%2 != 1:
        gb_x += 1
    if gb_y is None:
        gb_y = gb_x
    elif gb_y%2 != 1:
        gb_y += 1
    return cv2.GaussianBlur(img, (gb_x,gb_y),gb_s)
```

Esta escala indica el valor de los posibles píxeles en blanco y negro, en este caso 256.

```
escala = 256
```

Posteriormente, generamos las ventanas asociadas a las imágenes de entrenamiento.

```
lista_pruebas_vectores = generar_pruebas(img,
distancia,angulo,escala,img.shape[0],img.shape[1],
n)
```

Una vez obtenidas la matriz de GLCM y calculados los vectores de estadísticos de segundo orden, los etiquetamos, asociando la clase de la imagen de entrenamiento a la que corresponden los vectores de estadísticos y posteriormente poder entrenar los clasificadores con estos vectores.

```
lista_pruebas_vectores_etiquetados =
etiquetar(lista_pruebas_vectores, etiqueta)
```

Con esta función obtendremos una lista que tendrá

los vectores de prueba, con su respectiva etiqueta.

```
return lista_pruebas_vectores_etiquetados
```

Para la creación de nuestro set de pruebas utilizamos un for, el cual recorrerá la carpeta que contiene nuestras imágenes de entrenamiento, en este caso se realizará una validación, esta validación consta de comparar el nombre de las imágenes, en este caso si el nombre la imagen termina en S, se le pondrá la etiqueta 1, sin embargo si la imagen termina en N, se le pondrá la etiqueta 0.

```
for imagen in os.listdir(PATH):
    if imagen.endswith("S.png"):
        texturas_url.append( (PATH+imagen, 1) )
```

Posteriormente, se generan todas las muestras con sus respectivas etiquetas.

```
train_set = [sacar_pruebas(url,
distancia=distancia,
n=n,angulo=angulo,
etiqueta=etq)
for url, etq in texturas_url]
```

Generando un clasificador.

Para este proyecto ya no es necesario que se implementen de cero los clasificadores por lo que importamos las bibliotecas necesarias para los clasificadores, en este caso es el módulo MultinomialNB, neighbors de Sklearn y regresión logística.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
```

Primero encadenamos las pruebas para hacer solo una lista.

```
train_set_new = list()
for lista in train_set:
    train_set_new.extend(lista)
train_set = train_set_new
```

Agrupamos nuestros datos en vector de estadísticos de segundo orden y en etiqueta de textura.

```
X = np.array([*[x[0] for x in train_set]])
y = np.array([*[x[1] for x in train_set]])
```

Aplicamos los clasificadores con los datos de entrenamiento. Una vez obtenidos los conjuntos de entrenamiento, instanciamos los clasificadores y los entrenamos para poder realizar predicciones.

- Clasificador Bayesiano


```
clf = MultinomialNB()
clf.fit(X, y)
```

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

- Clasificador K-NN

```
neigh = KNeighborsClassifier(n_neighbors=4)
neigh.fit(X, y)
```
- Regresión logística.

```
LR = LogisticRegression(max_iter=10000)
LR.fit(X, Y)
```

Para poder realizar las pruebas con los clasificadores, hicimos la función llamada `generar_prediccion` esta función recibe 6 parámetros, el primer parámetro es la imagen a analizar, el segundo parámetro es la distancia que se utilizará para poder crear las ventanas, el tercer y cuarto parámetro es el tamaño de la imagen (X Y), el quinto parámetro es el ángulo que se utilizará para crear la matriz glcm, el sexto parámetro, llamado escala, es el número de posibles pixeles que se pueden ocupar, esto guiándonos a través de la escala de grises, el séptimo parámetro que recibe, es el clasificador con el que se realizarán las pruebas y por último la variable n que corresponde al tamaño de la ventana.

La función tiene una estructura similar a la función para generar pruebas, pero en este caso realiza los siguientes pasos, al inicio de la función, tendremos un if, en donde se validará si la variable n contiene algo, si esta variable, no contiene un dato, se realizará la siguiente operación para obtenerla.

```
n = distancia*2+1
```

La siguiente validación que se realizará, será el contenido de la variable `tamaño_y`, si esta variable es none, se le asignará el valor de `tamaño_x`.

Por lo tanto si se manda una distancia de 2, la ventana que obtendremos será de 5x5, para poder obtener las ventanas, se va recorriendo la matriz de la imagen, a partir del tamaño ingresado, posteriormente, cada ventana se guarda en una lista generada al inicio de la función.

```
for i in range(0,tam_x-n,distancia):
    for j in range(0,tam_y-n,distancia):
        matriz = imagen[i:i+n,j:j+n,:]
```

En este caso se realizará la matriz de GLCM para los tres diferentes canales de nuestra imagen, estas matrices resultantes se sumarán para obtener una sola matriz y al final, la función regresará las ventanas en una lista.

```
Vector_Prediccion=np.array([R_GLCM+G_GLCM+B_GLCM])
```

Después de obtener el vector con sus respectivos estadísticos de segundo orden, realizamos las predicciones para los vectores de la imagen de prueba.

```
Solucion=clasificador.predict(Vector_Prediccion)
```

Después de aplicar el clasificador, pintaremos los pixeles de la imagen a analizar, con el color correspondiente a

la clase que nos arrojó el clasificador, para poder observar el resultado.

```
nuevaImagen[i:i+n,j:j+n]=int(Solucion[0])
```

Generación de resultados.

Para mostrar los resultados en pantalla, se utilizó la función llamada predicción, esta función, recibe el modelo con el cual se generará la predicción, dentro de la función, se seguirá el siguiente flujo, primero se utilizará la función `generar_prediccion`, la cual nos regresará la predicción, posteriormente se mostrará la imagen original y la imagen de resultado, además se obtendrá la cuantificación en pixeles del sargazo, a partir de la siguiente línea de código, además del número total de pixeles clasificados como sargazo.

```
print("PIXELES CLASIFICADOS COMO SARGAZO:",pixeles_sargazo)
print("PORCENTAJE DE SARGAZO:",round((pixeles_sargazo/(imagen.shape[0]*imagen.shape[1]))*100 ,2),"%)
```

Resultados obtenidos

Los resultados mostrados a continuación, son producto del entrenamiento con solo dos clases, estas clases fueron las de sargazo y las de no sargazo, mostradas a continuación.



Figura 1. Ejemplo de máscara 1.



Figura 2. Ejemplo de máscara 2.

Resultados obtenidos.

Imagen 1.

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

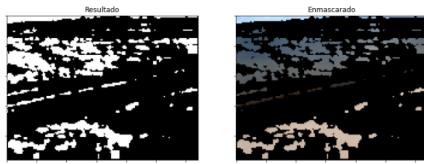


Figura 3. Resultados obtenidos con el clasificador de Bayes.

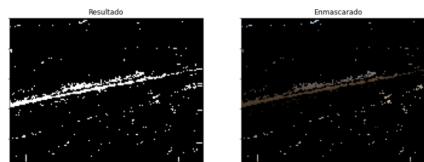


Figura 4. Resultados obtenidos con el clasificador de K-NN (4 clusters).

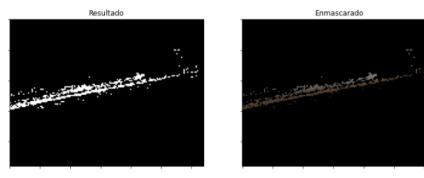


Figura 5. Resultados obtenidos con el clasificador de K-NN (2 clusters).

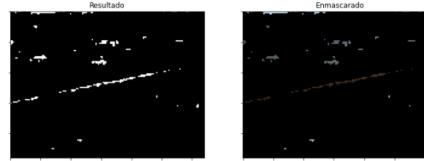


Figura 6. Resultados obtenidos con Regresión Logística.

Imagen 2.

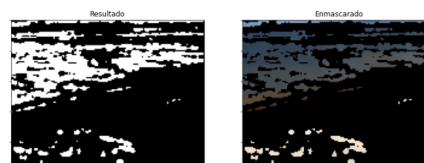


Figura 7. Resultados obtenidos con el clasificador de Bayes.

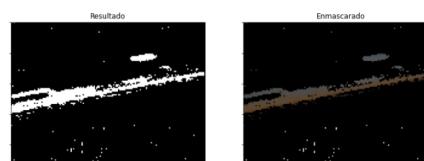


Figura 8. Resultados obtenidos con el clasificador de K-NN (4 clusters).

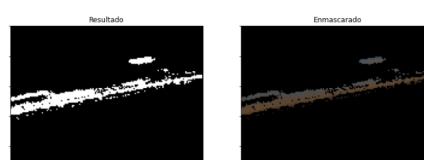


Figura 9. Resultados obtenidos con el clasificador de K-NN (2 clusters).



Figura 10. Resultados obtenidos con Regresión Logística.

Como se puede observar en los resultados, en los cuatro se obtuvieron buenos resultados, sin embargo, el que obtuvo mejores resultados fue el clasificado llamado K-NN, pero con 2 clustering.

Validación.

La validación cruzada o cross validation es un método para obtener la precisión de un algoritmo de clasificación o pronóstico y consiste en dividir el dataset de entrenamiento en k segmentos, donde k-1 segmentos fungirán como data de entrenamiento y el segmento k será para validar el modelo. En cada iteración se irá variando el segmento de prueba y con el resultado de cada uno de estos se obtendrá la precisión general del modelo.

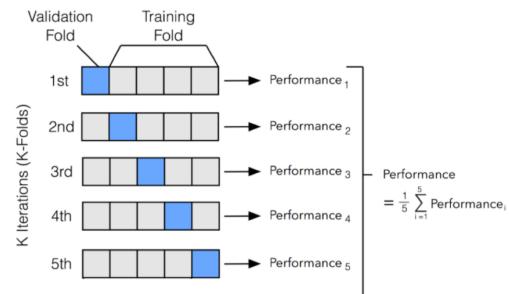


Fig. 11. Diagrama de validación cruzada.

La ventaja que ofrece este método de validación con respecto a otros es que la información no permanece “estática”, ya que la información de prueba y de entrenamiento va variando en comparación con un modelo que solo entrena con una parte de los datos y otra de pruebas sin variaciones. Por lo tanto, la precisión que ofrecerá la validación cruzada será ligeramente más elevada que la mencionada anteriormente.

Para utilizarla únicamente hay que importar la siguiente librería:

```
from sklearn.model_selection import
cross_val_score
```

A continuación se puede observar en la tabla las precisiones de los modelos obtenidas con una distancia entre píxeles de 8 y una ventana de 32 píxeles de tamaño.

Precisión Bayes	Precisión K-NN (4 clusters)	Precisión K-NN2 (2 clusters)	Regresión logística
68.91%	95.03%	95.88%	85.12%

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

Tabla 1. Score de los modelos generados en la Parte A.

Parte B SEGMENTACIÓN con normalización (Utilización de clasificadores)

Cuando hicimos la parte A notamos que había algunas imágenes más opacas que otras y al observar los resultados nos preguntamos si era posible que esta diferencia de iluminación afectaba a nuestros clasificadores. Por lo tanto, decidimos hacer una normalización de los colores de las imágenes con el objetivo de mejorar nuestros resultados.



Figura 12. Comparación entre imágenes opacas y claras del set de entrenamiento.

Para solucionar este problema se creó la función *normalizar* que se encarga de encontrar el píxel cuyo valor sea el más pequeño de cada fila y establecerlo como el origen, es decir, ese valor será “el nuevo cero” afectando al resto de los píxeles de la fila. De esta manera tendremos imágenes más opacas y homogéneas entre sí.



Figura 13. Comparación entre la imagen original (izquierda) y la misma imagen normalizada (derecha).

De igual manera las máscaras utilizadas en la parte A deben ser normalizadas para este nuevo análisis. Su normalización requirió la modificación de la función *sacar pruebas* descrita en la parte anterior y se apoya de una nueva función llamada *imágenes normalizadas*.

Esencialmente la función *sacar pruebas normalizadas* realiza el mismo procedimiento que la función *sacar pruebas*, pero la principal diferencia es que llama la función *imágenes normalizada* la cual recibe como parámetro a la imagen original y dentro el procedimiento también llama a sus respectivas dos máscaras (una donde se oculta al sargazo y otra donde se oculta todo menos el sargazo). Después de

aplicar el blur a la imagen original, la función detecta al píxel de valor más bajo tanto en las filas como las columnas para establecerlo como el nuevo origen. Así se tiene una normalización de todos los colores de la imagen.

Ahora bien, para normalizar las máscaras simplemente multiplicamos todos los píxeles que sean diferente de 0 en la máscara original con blur (Figura 13) con la imagen original normalizada (Figura 14) para obtener la máscara normalizada (Figura 15). Es posible notar que la región del sargazo en la nueva máscara es más amplia en comparación con la máscara original.



Figura 14. Máscara sin normalizar con blur.



Figura 15. Imagen normalizada de donde se extrajo la máscara normalizada.



Figura 16. Máscara normalizada de la parte B.

Esta función se aplica tanto a las máscaras que contienen el sargazo como en aquellas que lo ocultan.

Generando un clasificador y generación de resultados

Con el objetivo de comparar únicamente el efecto de normalización en las imágenes, se utilizaron los mismos

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

clasificadores descritos en la Parte A y no se les realizó ningún cambio en sus parámetros: Clasificador bayesiano, K-NN con 4 clusters, K-NN con 2 clusters y regresión logística.

De la misma forma se utilizó la función generar predicción para realizar la clasificación y desplegar la cantidad de píxeles clasificados como sargazo y su porcentaje en la imagen.

Resultados obtenidos

Imagen 1.

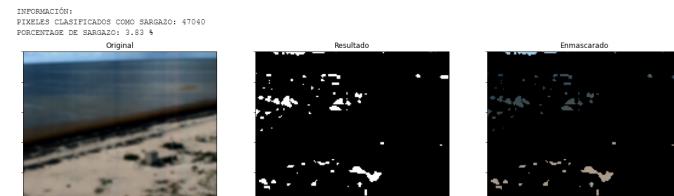


Figura 17. Resultados obtenidos con el clasificador de Bayes.

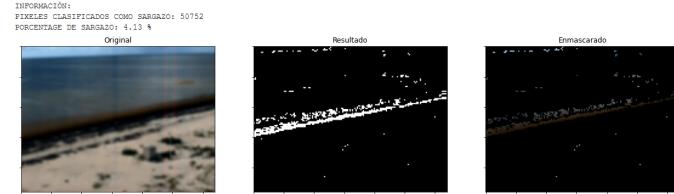


Figura 18. Resultados obtenidos con el clasificador de K-NN (4 clusters).

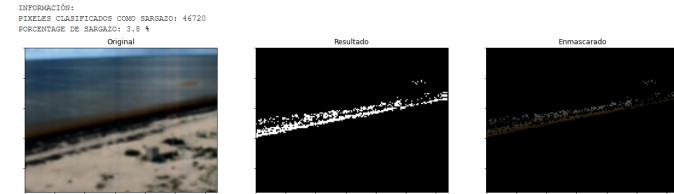


Figura 19. Resultados obtenidos con el clasificador de K-NN (2 clusters).

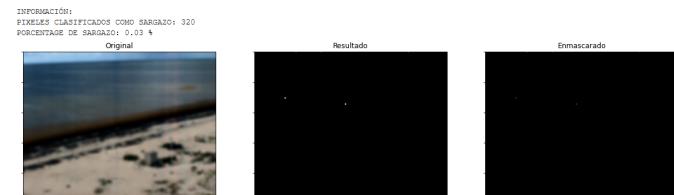


Figura 20. Resultados obtenidos con Regresión Logística.

Imagen 2.



Figura 23. Resultados obtenidos con el clasificador de Bayes.

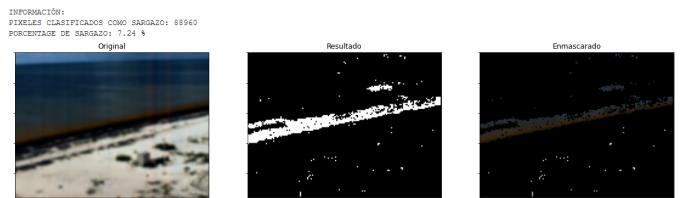


Figura 24. Resultados obtenidos con el clasificador de K-NN (4 clusters).

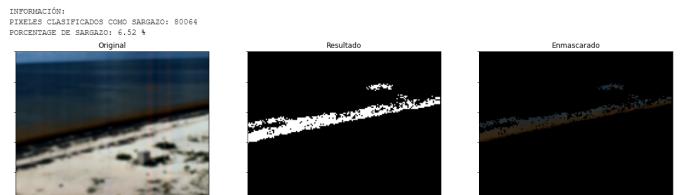


Figura 21. Resultados obtenidos con el clasificador de K-NN (2 clusters).

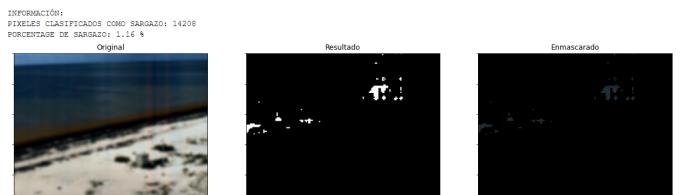


Figura 22. Resultados obtenidos con Regresión Logística.

Validación.

Para la validación se volvió a utilizar el método de validación cruzada y se obtuvieron los resultados de la Tabla 2.

Precisión Bayes	Precisión K-NN (4 clusters)	Precisión K-NN2 (2 clusters)	Regresión logística
78.44%	95.36%	96.19%	88.09%

Tabla 2. Score de los modelos generados en la Parte B.

Es posible notar que hubo una mejora considerable para el modelo de clasificación bayesiana; para los modelos de k vecinos más próximos se tuvo un resultado aproximado, sin cambios significativos y finalmente para el modelo de regresión logística se tuvo un aumento de tres puntos porcentuales en su precisión.

Sin embargo, observando los resultados de las imágenes 1 y 2 de regresión logística de la Parte B tiene una clasificación totalmente deficiente en contraparte con sus pares de la Parte A.

Parte C Redes Neuronales (Utilización de clasificadores)

Esta parte del proyecto fue más sencilla que la utilizada con GLCM. Aquí, utilizaremos los píxeles tal cual pertenecen a la clase sargazo. Para poder obtener los píxeles de la clase sargazo, utilizamos las siguientes líneas de código.

```
re_sar = sar.reshape((-1, 3))
lista_sar = re_sar[np.sum(re_sar, axis=1) != 0, :]
```

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

```
lista_etiquetada = etiquetar(lista_sar, 1)
```

Pasando una matriz de píxeles RGB a un vector de píxeles RGB. Después, se filtran los píxeles que solo son negros y se etiqueta la lista con 1.

Para la clase no sargazo, no se ocupan funciones similares, pero se limita el número de píxeles de la clase al doble de los píxeles que hay para la clase sargazo, esto para no sobreajustar o afectar al modelo.

El modelo de la red neuronal se genera con las siguiente líneas de código.

```
model = Sequential()
model.add(Dense(20, input_dim=3,
activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Teniendo una arquitectura de 2 capas ocultas, con capas de 20, 10, 5 y 1 neurona. Las capas utilizan una función de activación, a excepción de la última, la cual utiliza una función sigmoide para obtener una probabilidad. La salida de la red se clasifica con un umbral de 0.5, siendo los valores mayores a 0.5 sargazo, y menores, no sargazo.

Para el modelo, se utilizó un optimizador Adam, con un learning rate de 0.002 y unas bethas de 0.9 y 0.999 respectivamente. Se ocupó también una función de costo basada en la entropía cruzada y una función de precisión binaria para evaluar el sistema.

```
optimizador=
keras.optimizers.Adam(learning_rate=0.002,
beta_1=0.9)
model.compile(loss='binary_crossentropy',
optimizer=optimizador,
metrics=['binary_accuracy'])
```

El sistema se entrenó en 100 épocas, con un tamaño de batch de 1000 imágenes por iteración. La predicción de las nuevas imágenes también fue mucho más rápida, ya que se pueden generar predicciones de manera paralela con todos los píxeles, y redimensionando al final.

```
re_img = imagen.reshape((-1, 3))
prediccion = clasificador.predict(re_img)
prediccion =
prediccion.reshape((imagen.shape[0], imagen.shape[1]))
```

Generación de resultados.

Se utilizó la función de evaluación para evaluar el sistema respecto a los datos de entrenamiento. La evaluación arrojó un valor del 99.94% de precisión.

```
16141/16141 [=====] - 53s 3ms/step - loss: 0.0025 - binary_accuracy: 0.9994
binary_accuracy: 99.94%
```

Figura 24. Resultados obtenidos de la evaluación de la red neuronal con precisión binaria.

Resultados obtenidos

Los resultados mostrados a continuación, son producto del entrenamiento con solo dos clases, estas clases fueron las de sargazo y las de no sargazo. En las siguientes figuras podemos observar los resultados arrojados.

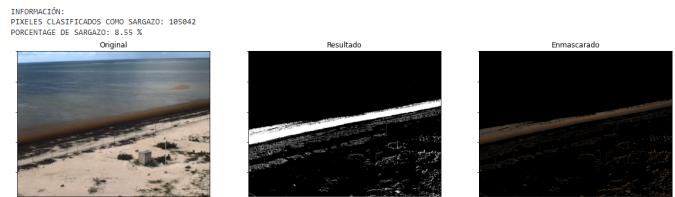


Figura 24. Resultados obtenidos de la evaluación de la red neuronal con precisión binaria.

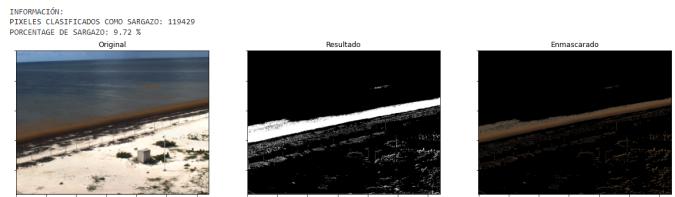


Figura 25. Resultados obtenidos con la red neuronal para la imagen 1.



Figura 26. Resultados obtenidos con la red neuronal para la imagen 2.

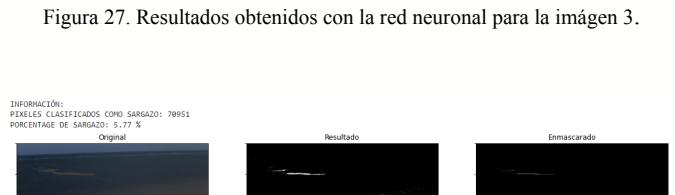


Figura 27. Resultados obtenidos con la red neuronal para la imagen 3.

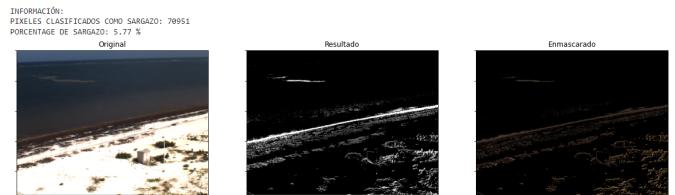


Figura 28. Resultados obtenidos con la red neuronal para la imagen de prueba.

IV. CONCLUSIONES

Esta práctica fue muy interesante, ya que al finalizar la práctica pudimos obtener la segmentación de sargazo en diferentes imágenes, para ello utilizamos tres métodos, para poder observar en cuál de los tres métodos, la segmentación se realizaba de mejor manera, sin embargo al comenzar con el primer método tuvimos un error, pues no sabíamos el número de máscaras a utilizar, en el primer intento utilizamos cuatro máscaras, sin embargo nos dimos cuenta que el entrenamiento era demasiado tardado, por lo que cambiamos a dos máscaras, la primer mascara contenía el sargazo y la segunda máscara

RECONOCIMIENTO DE PATRONES. PROYECTO FINAL

contenía los demás elementos que tenía la imagen, después de entrenar nuestros clasificadores con estas imágenes, pudimos observar que el clasificador KNN, era el que nos arrojaba mejores resultados, por lo que cambiamos el número de clusters que tenía el clasificador, para cambiar de 4 a 2 clusters, después de ver el análisis, pudimos observar que el clasificador con 2 clusters, era el que mejores resultados arroja.

Otro de los problemas visto en la implementación de este clasificador, fue que en una de las pruebas realizadas, decidimos quitar la entropía en los estadísticos de segundo orden, pero los resultados obtenidos al final, no mejoraba, por lo que procedimos a dejar la entropía en los estadísticos de segundo orden.

Un problema constante que tuvimos desde prácticas pasadas, fue que al utilizar muchas estructuras de datos complejas, la memoria se volvía un problema. En este proyecto, aplicamos la utilización de la instrucción del de python para tratar de mantener los niveles de utilización de la memoria RAM bajos y que el servicio de Google Colab no colapsara.

Por último y después de ver los diferentes métodos, pudimos observar como fallaban la segmentación, por lo que se decidió realizar un nuevo método y otras máscaras, en estas máscaras, solo se tomó en cuenta el sargazo que se encontraba dentro del mar.

Red Neuronal

El sistema arrojó resultados prometedores. El único inconveniente fue que clasificaba todos los pixeles de color café como sargazo. Esto no es muy malo, ya que de todas las aproximaciones, es la que mejor clasifica las manchas de sargazo en el mar. A simple inspección, es el sistema que mayor precisión tuvo, junto con el de knn arrojado en las primeras aproximaciones usando GLCM.

Algo que podría mejorar el sistema, es la presencia de más datos, y de la posible utilización de funciones tangente hiperbólica como función de activación, esto buscando menos linealidad en el sistema.

REFERENCIAS

- [1] Direccion, M. (2020, 19 junio). Conoce qué es el sargazo y su función en el océano. Acuario Michin. <https://acuariomichin.com/conoce-que-es-el-sargazo-y-su-funcion-en-el-oceano/> (accessed Dic. 09, 2021).
- [2] Roman, V. (2019, 29 abril). “Algoritmos Naive Bayes: Análisis de texturas - MATLAB & Simulink - MathWorks España. (s. f.). mathworks. <https://es.mathworks.com/help/images/texture-analysis-1.html> (accessed Dic. 09, 2021).
- [3] Glosario: Validación de métodos. (s. f.). greenfacts. <https://www.greenfacts.org/es/glosario/tuv/validacion.htm> (accessed Nov. 18, 2021).
- [4] El algoritmo K-NN y su importancia en el modelado de datos | Blog. (s. f.). Merkle. <https://www.merkleinc.com/es/es/blog/algoritmo-knn-modelado-datos> (accessed Dic. 09, 2021).
- [5] kmeans. (s. f.). unioviedo.

https://www.unioviedo.es/comppnum/laboratorios_py/kmeans/kmean_s.html (accessed Dic. 09, 2021).

Rodrigo, J. A. (s. f.). “Regresión logística simple y múltiple. Ciencia de Datos.”

https://www.cienciadadedatos.net/documentos/27_Regresion_logistica_simple_y_multiple#Regresi%C3%B3n_log%C3%ADstica_m%C3%A1ltiple (accessed Dic. 09, 2021).

El modelo de redes neuronales. (s. f.). IBM.

<https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=netsworks-neural-model> (accessed Dic. 09, 2021).

D. (2021, 12 enero). Optimización de descenso de gradiente de Code Adam desde cero. Top Big Data.

<https://topbigdata.es/optimizacion-de-descenso-de-gradiente-de-code-adam-desde-cero/> (accessed Dic. 09, 2021).

Definición de entropía cruzada. (s. f.). LOKAD.

<https://www.lokad.com/es/definicion-de-entropia-cruzada> (accessed Dic. 09, 2021).