Albert Ripoll Batlló y Wenjie Jin

Máster Big Data La Salle, Barcelona

Práctica Data Mining

Mar 8, 2025

Índice

Introducción al problema **2**
- Antecedentes 2
- Objetivo del proyecto 2

Recopilación de los datos **6**

Visualización **7**
- 1- Formas de los audios 7
- 2- Espectrogramas 7
- 3- Elección de frame size y hop length 8

Extracción de las características **12**
- 1- Descripción de las características obtenidas 12
- 2- Extracción de características 15
- 3- Estandarización de dimensiones 17

Balanceo de los datos **18**

Modelos de machine learning **19**
- **RF (Random Forest)** **19**
- **KNN (k-Nearest Neighbors)** **24**
- **CNN (Convolutional Neural Networks)** **27**

Evaluación de los modelos **32**
- **RF (Random Forest)** **32**
- **KNN (k-Nearest Neighbors)** **38**

Conclusiones **42**

Introducción al problema

Antecedentes

En los últimos 30 años, el número de barcos ha aumentado de forma exponencial. Sus emisiones de ruido de baja frecuencia (50-1000 Hz) coinciden con el rango acústico utilizado por especies marinas para comunicarse, generando un efecto de enmascaramiento sonoro con graves consecuencias ecológicas[1].

Entre las principales víctimas de este problema se encuentra la población de ballenas francas del Atlántico norte, que ha caído a menos de 350 ejemplares debido a la exposición crónica al ruido. En el Mediterráneo, la eficacia de caza de las marsopas se ha reducido en un 60% por daños auditivos, afectando su supervivencia.

Sin embargo, no solo los grandes mamíferos marinos se ven afectados. Organismos más pequeños, como los *Oikopleura dioica*, deuteróstomos invertebrados esenciales para la red trófica marina, también sufren el impacto del ruido submarino. Estas diminutas criaturas desempeñan un papel crucial en el ciclo del carbono y la dinámica del plancton, y su sensibilidad al ruido generado por el tráfico marítimo podría tener efectos en cascada sobre los ecosistemas oceánicos.

Dada la magnitud de este problema, es fundamental desarrollar herramientas que permitan evaluar y mitigar el impacto del ruido submarino sobre la vida marina. A pesar de las directrices de la Organización Marítima Internacional (IMO), las tecnologías tradicionales de monitoreo dependen de análisis manuales, son ineficientes y fallan en escenarios complejos (como múltiples barcos o interferencias ambientales), lo que exige soluciones inteligentes.

Objetivo del proyecto

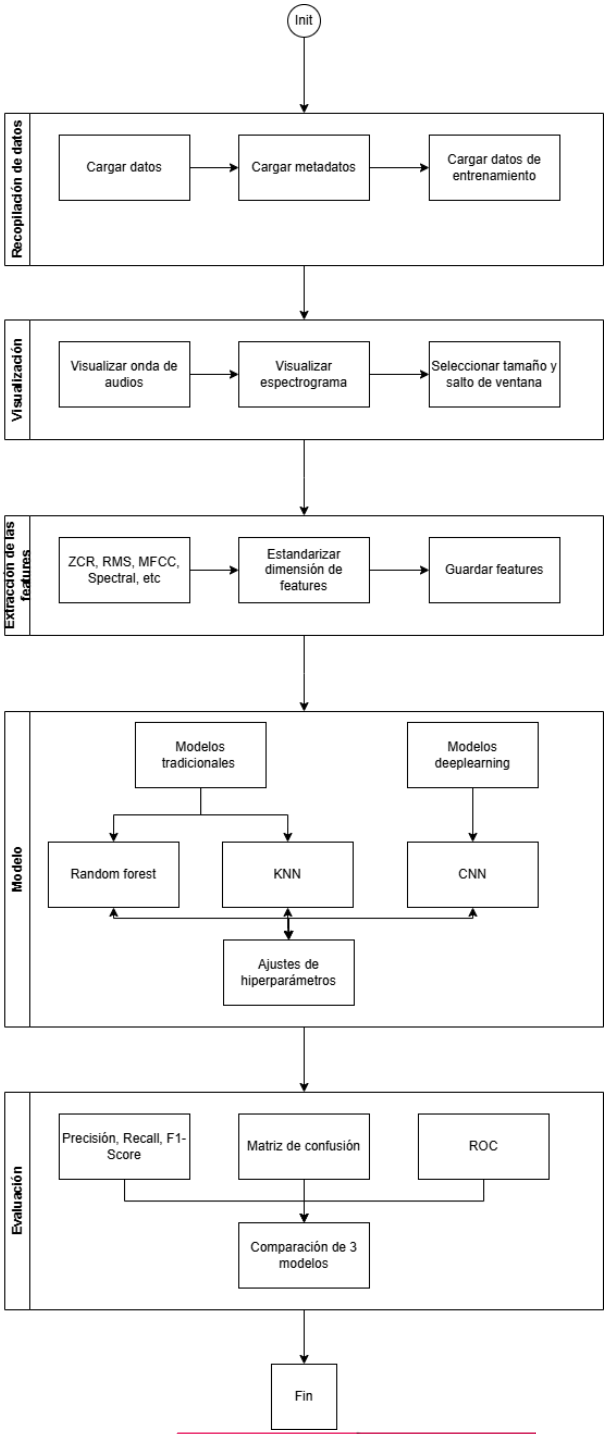
Delante de estos desafíos de ruido submarino, surge el proyecto DeuteroNoise con el objetivo de evaluar el impacto del tráfico marítimo sobre los Oikopleura Dioica, unos deuterostomos invertebrados marítimos. Como pieza fundamental de DeuteroNoise, el presente trabajo tiene como objetivo desarrollar una solución basada en la tecnología de minería de datos con los algoritmos de machine learning para analizar y clasificar los tipos de barcos en función de audios proporcionados por DeuteroNoise. Lo que se aspira en este proyecto es crear un modelo de ML que clasifique audios de forma inteligente y precisa, identificando cada barco como lo haría un experto humano, pero de forma más rápida y eficiente.

---

[1] Fuente: Rodrigo Saura, F. J. (2014). *La Contaminación Acústica Submarina: Fuentes e Impacto Biológico*. Sociedad Anónima de Electrónica Submarina (SAES). Recuperado de https://electronica-submarina.com/wp-content/uploads/La-Contaminacion-Ac%C3%BAstica-Submarina-Fuentes-e-Impacto-Biologico-SAES_web.pdf

Explicación del pipeline

El esquema de pipeline consiste en las siguientes etapas:



Etapa 1 - Recopilación de los datos

Se omiten las etapas de eliminación de ruido y segmentación manual ya que los audios han sido previamente procesados y segmentados en 5 segmentos de los 4 segundos de mayor intensidad sonora.

- **Carga de datos:** Se importan los archivos de audio, organizados en cuatro carpetas principales cada una representando una clase diferente de barco. Cada carpeta consta de múltiples registros organizados en subcarpetas que contienen cinco audios grabados por el mismo dispositivo.
- **Carga de metadatos:** Se incorporan archivos con información adicional relevante, como la fecha de grabación, la distancia del vehículo marítimo del hidrófono y la duración del audio.

Etapa 2 - Visualización

Antes de la extracción de características, se realiza un análisis exploratorio de los datos:

- **Visualización de la onda del audio:** Se representa la señal en el dominio temporal para comprender mejor su estructura.
- **Visualización del espectrograma:** Se examina la distribución de frecuencias a lo largo del tiempo mediante espectrogramas.

- **Selección del tamaño y el salto de ventana:** Se establecen los parámetros de segmentación del audio para la extracción de características, determinando la resolución temporal y espectral del análisis.

Etapa 3 - Extracción de las características

- **Extracción de las características**: Se calculan distintos atributos del audio, tales como:
  - **Tasa de cruces por cero (ZCR)**: Indica la frecuencia con la que la señal cambia de signo.
  - **Energía RMS**: Representa la intensidad del sonido.
  - **Coeficientes cepstrales en frecuencia Mel (MFCC)**: Permiten capturar información relevante sobre el timbre del sonido.
  - **Características espectrales**: Incluyendo el centroide espectral, la propagación espectral, entre otros.
- **Estandarización de las dimensiones**: Se ajustan las características a una forma consistente para su posterior uso en los modelos, aplicando normalizaciones y escalados.
- **Almacenamiento de las características**: Se guardan los valores extraídos para futuras etapas del proceso.

Etapa 4 - Balanceo de datos

- **Upsampling**: Se aplica el algoritmo SMOTE para aumentar la cantidad de muestras de clases minoritarias y mejorar el equilibrio del conjunto de datos.

Etapa 5 - Modelos de machine learning

Como bien se define en el enunciado de la práctica, se implementan 3 modelos diferentes para demostrar el rendimiento sobre la problemática de clasificación de audios.

- **Modelos Ensemble:** Se implementa un modelo ensemble como:
  - **Random Forest**: Un modelo basado en árboles de decisión.
  - **K-Nearest Neighbors (KNN)**: Un algoritmo basado en la similitud entre instancias.
- **Modelos de aprendizaje profundo**: Se implementa una Red Neuronal Convolucional (CNN) para capturar patrones complejos en los datos de audio.

- **Ajuste de los hiper parámetros**: Se optimizan los modelos ajustando distintos parámetros para mejorar su rendimiento.

Etapa 6️⃣ - Evaluación de los modelos

- **Precisión, Recall y F1-Score**: Se utilizan estas métricas clave para evaluar la calidad de la clasificación.
- **Matriz de confusión**: Se visualizan los errores de clasificación y las tasas de falsos positivos y negativos.
- **Curva ROC**: Se evalúa la capacidad discriminativa del modelo en diferentes umbrales de decisión.
- **Comparación entre modelos**: Se comparan los resultados de los distintos enfoques utilizados para seleccionar la mejor opción.

Etapa 7️⃣ - Fin de Pipeline

Una vez completadas todas las etapas anteriores, se procede con la redacción de la memoria del proyecto analizando los resultados obtenidos, extrayendo conclusiones y almacenando los modelos con mejor rendimiento.

Recopilación de los datos

En el proyecto DeuteroNoise, se han recopilado datos dejando hidrófonos registrando sonidos bajo el agua durante horas o días y se han cruzado los datos de los hidrófonos con datos de localización de los barcos para relacionar el sonido con el barco del cual procede. Se dispone de los siguientes datos:

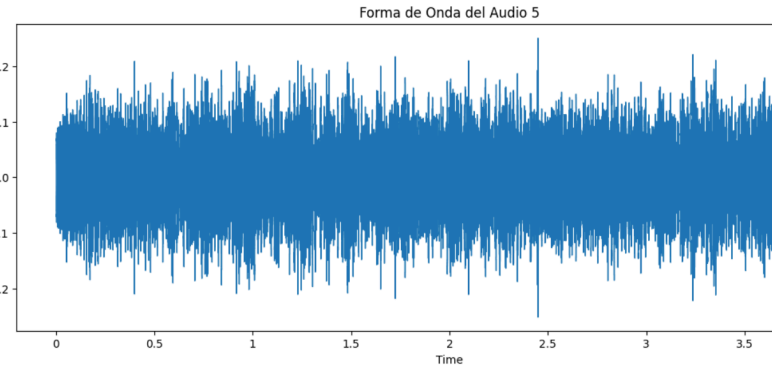
- Conjunto de datos grande con 20 GB de datos.
- Subconjunto de los datos anteriores de 1 GB que contiene hasta 5 fragmentos de 4 segundos de duración de un audio original seleccionando los fragmentos de más energía. Estos datos están ordenados según la siguiente distribución indicando el barco del cual procede el audio.
    - Cargo
        - Subcarpeta perteneciente a un audio que contiene 5 fragmentos de 4 segundos.
    - Passengership
        - Subcarpeta perteneciente a un audio que contiene 5 fragmentos de 4 segundos.
    - Tanker
        - Subcarpeta perteneciente a un audio que contiene 5 fragmentos de 4 segundos.
    - Tug
        - Subcarpeta perteneciente a un audio que contiene 5 fragmentos de 4 segundos.
- Fichero de metadatos que contiene:
    - ID de la clase.
    - Nombre del barco.
    - Día de la grabación en formato YYYYMMDD.
    - Duración del audio en segundos.
    - Distancia del barco al sensor. Indicado con dos números pertenecientes al máximo y mínimo de distancia durante el tiempo de registro del audio.

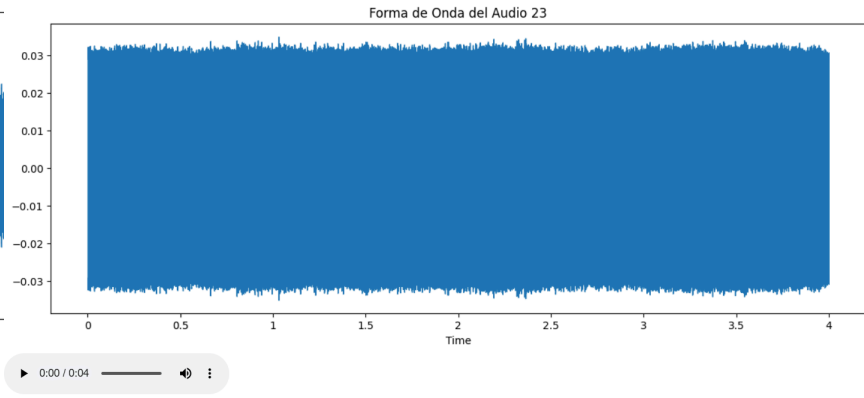
El presente trabajo se ha centrado en usar todos los 2816 audios del subconjunto de 1 GB. **code 19** No se ha entrado en sobrecargar el coste de computación usando los audios enteros. Tampoco se ha usado el análisis por subclases, nombres de barco o distancia del barco a pesar de pensar que la distancia fuente-registrador es un dato relevante a considerar.

Visualización

1- Formas de los audios

Se observa como son los datos de audio, qué forma tienen y como se escuchan.



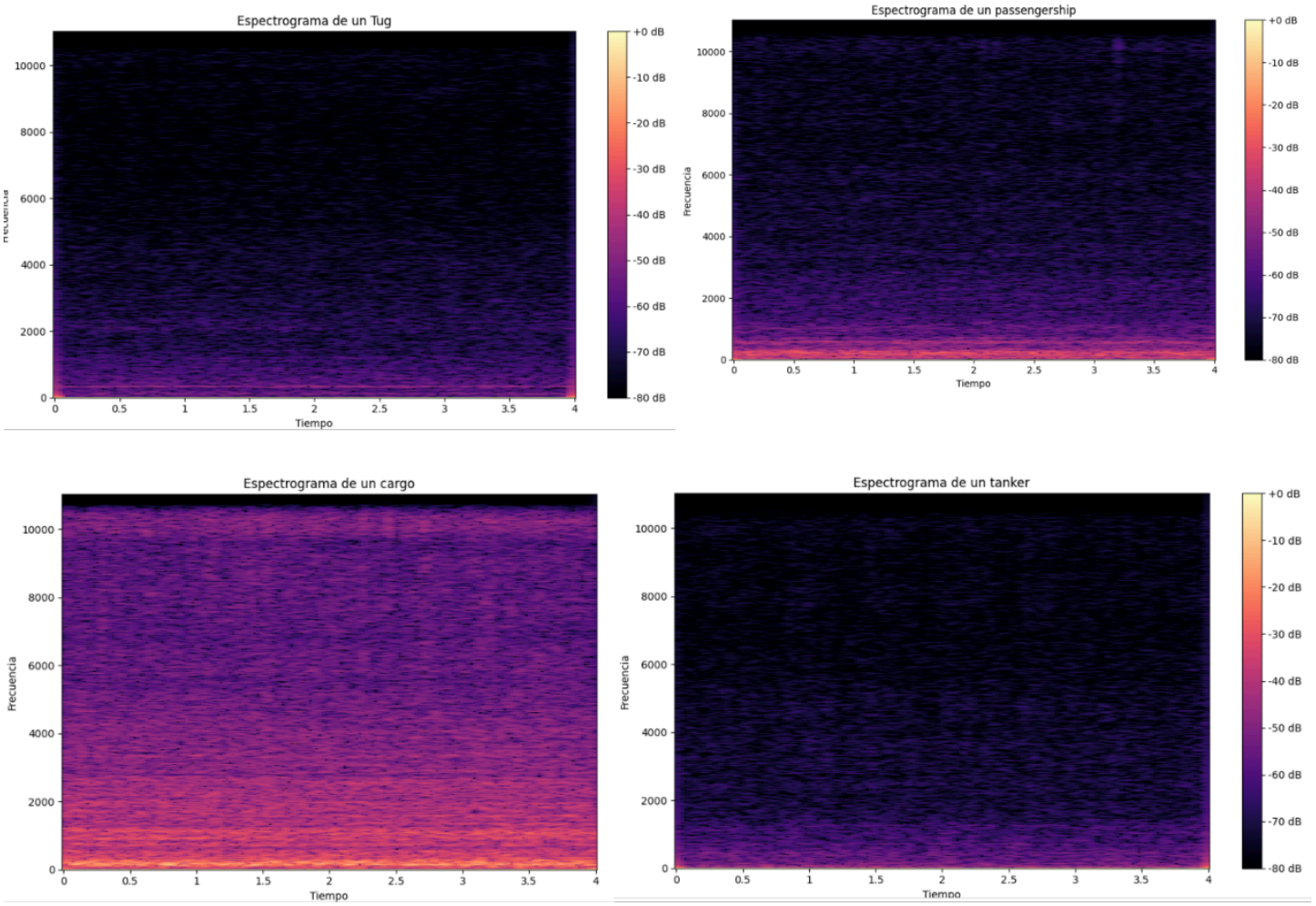


code 0

Todos los audios tienen una duración de 4 segundos. Los audios son de sonidos de barcos, registrados debajo del mar. Las frecuencias de los sonidos son frecuencias bajas (sonidos graves). El análisis de características del audio se hará por ventanas de muestreo del audio.

2- Espectrogramas

Se genera un espectrograma para cada tipo de audio para observar la distribución del sonido en frecuencias, que confirma la afirmación de mayor frecuencia baja.



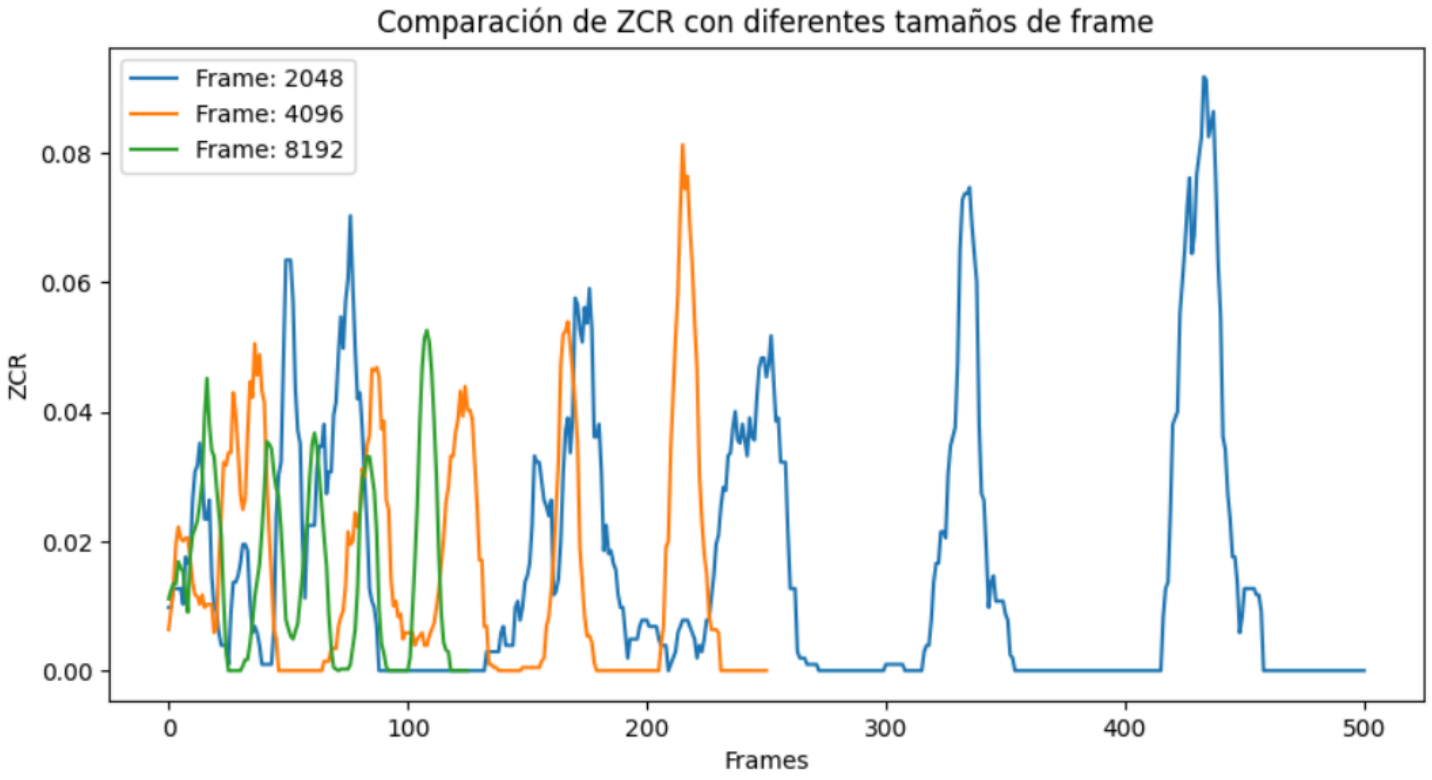
code 1

3- Elección de frame size y hop length

Ajuste de los valores de los audios

En este punto al intentar sacar las características de los audios se observa que ZCR siempre es 0 para cualquier audio. Hecho que no tiene lógica. Los audios deberían tener cruces por el cero. Al investigar, se observa que los valores de “y” al cargar los audios acostumbran a ser siempre positivos y nunca negativos, por eso no se detectan cruces por cero. Para solventar tal asunto se configura el audio para que el cero

siempre quede en el medio entre el máximo y el mínimo (`y = y - min(y) - (max(y) - min(y))/2`). Para elegir los tamaños de las ventanas adecuados, se compara el ZCR con los tamaños del frame y finalmente nos quedamos con la comparación de estas 3. **code 2**



Frame size

El frame size es la anchura o tamaño de la ventana. Determina cuántas muestras se usan para calcular las características. En cada ventana se calcularán todas las características. Aquí se pueden elegir valores altos o bajos, según el problema.

- Un valor alto (4096 o 8192), como se promedian más muestras, aporta peor resolución en tiempo. Sin embargo, proporciona mejor resolución en frecuencia, se ven más detalles en frecuencias bajas.
- Un valor bajo (512) mejora la resolución temporal detectando cambios rápidos en el tiempo, es decir frecuencias altas. Pero reduce la precisión y detalles en frecuencias bajas.

Hop length

El hop length o size es el espacio entre ventanas. Es decir, representa el número de muestras que se desplaza la ventana en cada paso. Si el hop size fuese igual al tamaño de ventana, no habría superposición de muestras entre segmentos. Pero es interesante que haya una superposición para tener un análisis más preciso. Se pueden elegir saltos altos o bajos según el problema.

- Un valor bajo (Frame_size/8 o Frame_size/4) da mejor resolución temporal pero aumenta el costo computacional ya que aumenta el número de datos a procesar.
- Un valor alto (Frame_size/2 o más) hace que el espectrograma sea más disperso en el tiempo. Es más rápido de calcular por lo que se disminuye el costo computacional.

Elección del frame size y hop length adecuado al problema

- Los sonidos de barcos típicamente suelen estar en frecuencias bajas (< 1000 Hz, a veces incluso < 300 Hz).[2] Dado que queremos alta resolución en frecuencia para distinguir detalles en buena resolución en esas frecuencias, es recomendable usar ventanas grandes (4096 o 8192).
- Los audios duran 4 segundos, por lo que necesitamos una buena distribución temporal. Teniendo unos datos con sampling Rate (sr) = 22050 Hz / 32000 Hz implica una frecuencia analizable de Nyquist entre 0 y 11025 Hz / 16000 Hz. [*Teorema de Nyquist: Una señal analógica de frecuencia máxima f_Max puede ser reconstruida completamente a partir de sus muestras si la tasa de muestreo es al menos el doble de f_Max*][3]. Es por eso que se intentará usar valores bajos de hop length siempre que el tiempo y costo computacional lo permita.

Por esas razones, los valores de frame size y hop length escogidos han sido los siguientes: **code 3**

```
Muestras por segundo =  32000 frames
```

[2] **Fundación Biodiversidad.** (2021). *Guía metodológica SILEMAR: Seguimiento del impacto del ruido submarino en el medio marino*. Fundación Biodiversidad. https://fundacion-biodiversidad.es/wp-content/uploads/2021/07/guia_metodologica_silemar.pdf
[3] **Baraniuk, R., et al.** (s.f.). *Teorema de Muestreo de Nyquist-Shannon*. LibreTexts Español. https://espanol.libretexts.org/Ingenieria/Se%C3%B1ales_y_Sistemas_(Baraniuk_et_al.)/10%3A_Muestreo_y_Reconstrucci%C3%B3n/10.02%3A_Teorema_de_Muestreo

```
Muestras por audio de 4s =  128000 frames

      HOP_LENGTH =  512 frames

      HOP_LENGTH =  0.016 segundos

      FRAME_SIZE =  4096 frames

      FRAME_SIZE =  0.128 segundos
```

De este modo, el análisis se hará según las características dentro de cada “frame” y se irán muestreando según cada “hop”.

Además, considerando la frecuencia de muestreo y los valores de longitud de onda y que cada audio tiene una duración de 4 segundos, obtendremos aproximadamente 173 valores para cada característica para cada audio, lo que representa 173 segmentos de tiempo.

```
4 * 22050 (sr) / 512 (hop_length) ≈ 173 muestras temporales
```

Extracción de las características

1- Descripción de las características obtenidas

Las características que son interesantes de analizar de los audios y que los hace distinguibles unos de los otros son las siguientes.

1.1 - Zero Crossing Rate (ZCR)

Es la cantidad de veces que una señal cruza a la amplitud 0 en un intervalo de tiempo. Es decir, en un diagrama amplitud - tiempo el ZCR son las veces que se cruza el eje horizontal. O sea, el número de ceros o raíces de la función amplitud(tiempo). Se calcula ZCR por ventanas.

1.2 - Root Mean Square (RMS)

En términos físicos tanto la energía (E), la potencia (P) como la intensidad (I) de una onda tridimensional dependen del cuadrado de la amplitud (A) como podemos demostrar: **demostración 4**

Por lo tanto, el Root Mean Square (RMS) es una medida estadística utilizada para cuantificar la cantidad de energía promedio presente en el sonido. Matemáticamente, se calcula de la siguiente manera:

$$\mathrm{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

x_i son los valores de la señal de audio.
N es el número total de muestras en la señal.

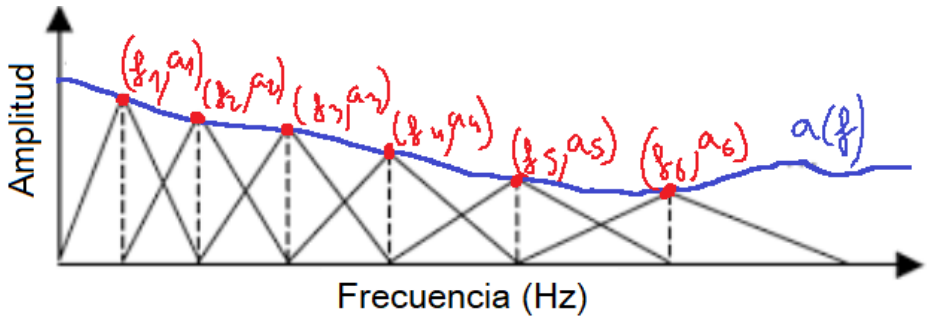
1.3 - Amplitude_mean_values

Otra forma de parametrizar la intensidad del sonido es fijándonos en las amplitudes que llega a dar el sonido en diferentes momentos de tiempo. Este parámetro nos informa de eso. Mira el valor máximo en cada ventana de análisis y devuelve la media de los máximos de amplitudes. Es una medida de la envolvente de la amplitud de la señal.

1.4 - Mel-Frequency Cepstral Coefficients (MFCC)

Es una representación compacta de las características espectrales del audio. Lo que son y cómo se calculan se podrían resumir en:

1) Mel-Scaling. Se aplica una escala de mel a la potencia espectral. Esta transformación no lineal tiene en cuenta como los humanos perciben la frecuencia del sonido y asigna mayor resolución en las frecuencias bajas (donde los humanos somos más sensibles) y menor resolución en las altas frecuencias. De este modo se asimilan las intensidades de las frecuencias acorde a lo que los humanos percibimos. Esta escala se hizo en base a tests experimentales porcentuales. A una muestra de personas se le presentaba una determinada frecuencia a una intensidad y la persona debería ponderar con un número a qué volumen percibía esta frecuencia. Con datos estadísticos de esas muestras se hizo la escala de mel.
2) Transformada de Fourier de tipo coseno discreta (DCT). Se realiza una transformación de la señal del dominio del tiempo al dominio de la frecuencia (transformada de Fourier) pero solo utilizando la serie de coseno, que reduce la redundancia de los coeficientes espectrales, hecho que ayuda a comprimir la cantidad de información. La DCT es una transformada discreta lo que significa que hace la transformación mediante un sumatorio en vez de hacerlo mediante una integral, este hecho aumenta la rapidez de computación.
3) Selección de coeficientes. Mediante un filtro triangular aplicado a una ventana de frecuencias en la función Amplitud - Frecuencia, a(f), se determinan los coeficientes. Esa ventana de frecuencias no es un intervalo constante, es un intervalo logarítmico de tal forma que se determinan más coeficientes en frecuencias más bajas. De estos coeficientes se seleccionan los 13 primeros coeficientes (a_k) de la frecuencia como características.



El trazado de líneas con la función a(f) determina unos puntos de frecuencia f_k, a_k para k=1,...,13. Donde a_k son los coeficientes mfcc seleccionados.

1.5 - Delta 1 y Delta 2

Son características que indican cómo cambian los coeficientes MFCC a lo largo del tiempo. Delta1 es el cambio de primer orden y delta2 el cambio de segundo orden.

1.6 - Spectral Centroid

Proporciona información sobre en qué frecuencias está centrada la señal, o sea, su tonalidad. Representa la frecuencia promedio o centro de gravedad del espectro de frecuencia. Matemáticamente se calcula haciendo el promedio ponderado según la energía de las frecuencias presentes en el espectrograma o transformada de Fourier.

$$\text{Centroid} = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

donde *x(n)* es el peso de energia de la frecuencia de la caja *n* y *f(n)* la frecuencia central de esa caja

1.7 - Spectral Flux

Describe la variación de energías del espectro entre segmentos consecutivos en el tiempo. En otras palabras, representa cómo varían las energías asociadas a cada frecuencia de una ventana de tiempo a otra.

1.8 - Spectral Bandwidth

Describe la anchura promedio de la distribución de frecuencias. Indica cuán amplio es el rango de frecuencias que abarca la señal promediado para todo el tiempo de audio.

2- Extracción de características

Con los audios cargados en un dataframe con columnas ship, audio_path, date, metadata_path y definidos los valores de hop_length y frame_size, se ha procedido a explorar los audios para extraer las características. La estructura ha sido: (**code 17** con funciones auxiliares en **code 21**)

1- Definir listas vacías.

2- Cargar cada audio con la ruta.

3- Aplicar la función para extraer las características y añadirlas a la lista.

4- Devolver un dataframe con las características.

2.1 - Zero Crossing Rate (ZCR)

Se aplicó `zcr = lb.feature.zero_crossing_rate(y=y)` **code 21**

2.2 - Root Mean Square (RMS)

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

x_i son los valores de la señal de audio.
N es el número total de muestras en la señal.

Al programarlo, se define la función `def root_mean_squared` que realiza el cálculo anterior sumando los valores de la señal de cada ventana (de i hasta i+frame_size) iterando *i* según el paso definido por hop_length. Se devuelve el valor de RMS de cada ventana. **code 21**

2.3 - Amplitude_mean_values

Al programarlo, se accede a cada ventana de valores (`for i in range(0, len(signal), hop_length): window = signal[i:i+frame_size]`) y se agrega el máximo de la amplitud de cada ventana en una lista. Después, se devuelve la media de esos máximos como medida de la envolvente de la amplitud de la señal. **code 21**

2.4 - Mel-Frequency Cepstral Coefficients (MFCC)

Diversos estudios y prácticas estándar[4] en el campo han establecido que utilizar 13 coeficientes MFCC es una elección adecuada para el análisis de audio. Esta elección se basa en que los primeros coeficientes capturan la envolvente espectral general de la señal, mientras que los coeficientes superiores suelen representar detalles más finos que pueden no ser esenciales para aplicaciones de aprendizaje automático y, en algunos casos, pueden introducir ruido o redundancia en el análisis. Por ese motivo se extraen 13 coeficientes:

```
mfcc = lb.feature.mfcc(y=y, n_mfcc=13, sr=sr)
```
code 21

2.5 - Delta 1 y Delta 2

Se aplicó: `delta1_mfcc = lb.feature.delta(mfcc, mode='nearest')`

```
delta2_mfcc = lb.feature.delta(mfcc, mode='nearest', order=2)
```
code 21

2.6 - Spectral Centroid

Se aplicó: `spectral_centroid = lb.feature.spectral_centroid(y=y, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)` **code 21**

2.7 - Spectral Flux

Se aplicó: `spectral_flux = lb.onset.onset_strength(S=lb.feature.melspectrogram(y=y, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH))` **code 21**

2.8 - Spectral Bandwidth

Se aplicó: `spectral_bandwidth = lb.feature.spectral_bandwidth(y=y, sr=sr, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)` **code 21**

---

[4] Meza Ruiz, I. V. (s.f.). *Coeficientes Cepstrales en la Frecuencia Mel (MFCC)*. Recuperado de https://turing.iimas.unam.mx/~ivanvladimir/posts/mfcc/
Revista Espacios. (2017). *Algoritmo de reconocimiento de comandos de voz basado en técnicas no supervisadas*. Recuperado de https://www.revistaespacios.com/a17v38n17/17381704.html

3- Estandarización de dimensiones

El resultado de extraer las características se compila en un dataframe de 2816 filas correspondiente a los audios analizados por 44 características analizadas (ZCR, RMS, Spectral_centroid, Spectral_flux, Spectral_Bandwidth, 13 coeficientes de MFCC, 13 coeficientes de Delta1, 13 coeficientes de Delta2). Cada celda contiene un vector de 173 valores correspondiente a las 173 ventanas que se analizaron a lo largo de los 4 segundos.

zcr	rms	spectral_centroid	spectral_flux	spectral_bandwidth	mfcc_1	mfcc_2	...
[0.05859375, 0.09130859375, 0.1298828125, 0.13...	[0.00299726898813614, 0.002886299138930153, 0....	[3000.1459062049735, 2888.3346905304343, 2825....	[0.0, 0.0, 0.0, 0.0021682892, 0.0011012929, 0....	[3180.845205597498, 3098.4059707293595, 3022.5...	[-427.7843322753906, -394.3990783691406, -393....	[71.79420471191406, 82.72506713867188, 85.8179...	...
[0.07470703125, 0.11572265625, 0.1484375, 0.14...	[0.0030928133342576587, 0.0030505756178857784,...	[3270.3577110126685, 3150.378608300146, 3039.3...	[0.0, 0.0, 0.0, 0.002216533, 0.0010595913, 0.0...	[3362.188968631459, 3283.4067408270935, 3179.1...	[-416.9086608886719, -385.4869689941406, -388....	[70.04466247558594, 77.05561828613281, 79.4091...	...
[0.0283203125, 0.0615234375, 0.09765625, 0.137...	[0.0034913358128646605, 0.0031674555340595934,...	[2813.411030672151, 2811.2522878102027, 2828.7...	[0.0, 0.0, 0.0, 0.001591587, 0.0010055545, 0.0...	[3130.6390975781774, 3059.944464739985, 2997.6...	[-418.4742736816406, -386.15521240234375, -387...	[79.3499755859375, 78.3955307006836, 79.265975...	...

Como valores de entrada de los modelos de aprendizaje automático que se puedan estandarizar, normalizar y después entrenar, es necesario que los valores de entrada tengan un formato concreto. Cada celda del dataframe no puede ser un vector de valores si no que tiene que ser un valor único. Por esa razón, en este paso se separan las 44 características en 173 valores individuales cada uno. **code 18**

zcr_1	zcr_2	zcr_3	zcr_4	zcr_5	zcr_6	zcr_7	...	delta2_mfcc_13_164
[0.05859375]	[0.09130859375]	[0.1298828125]	[0.13232421875]	[0.1484375]	[0.1552734375]	[0.16357421875]	...	[0.48696044087409973]
[0.07470703125]	[0.11572265625]	[0.1484375]	[0.1455078125]	[0.1435546875]	[0.15185546875]	[0.14697265625]	...	[0.1917257457971573]
[0.0283203125]	[0.0615234375]	[0.09765625]	[0.1376953125]	[0.1572265625]	[0.158203125]	[0.1591796875]	...	[0.33973345160484314]

Balanceo de los datos

Una vez extraídas las características, se identificó un problema de desbalance en las clases, lo que puede y parece que genera predicciones sesgadas hacia la clase mayoritaria. Dado que el entrenamiento se realiza sobre los atributos derivados de los audios, se optó por aplicar el ajuste de desbalanceo después de la etapa de extracción de características.

Para solventar este problema, se utilizó SMOTE (Synthetic Minority Over-sampling Technique), que genera muestras sintéticas de las clases minoritarias para equilibrar la distribución del dataset. Esta estrategia contribuye a mejorar la equidad en las predicciones y, en consecuencia, la capacidad del modelo para generalizar de forma adecuada. **code 20**

Modelos de machine learning

RF (Random Forest)

La elección del algoritmo Random Forest para el procesamiento del conjunto de datos de audios de submarinos se fundamenta en su robustez y eficiencia en el manejo de datos tabulares de alta dimensionalidad, tales como los derivados de características acústicas (por ejemplo, MFCC, ZCR, RMS y parámetros espectrales). Este algoritmo de ensamble, que integra múltiples árboles de decisión, contribuye a la reducción de la varianza y mejora la capacidad de generalización en entornos con alto nivel de ruido o variabilidad. Además, Random Forest permite obtener medidas de importancia para cada característica, lo que facilita la interpretación de los resultados, y destaca por su eficiencia computacional y su relativa facilidad de implementación.

Concepto del Random Forest

El Random Forest es un modelo de aprendizaje automático que se basa en la combinación de distintos árboles de decisión (*machine learning tree decision ensemble*). Al combinar distintos árboles, las predicciones son más robustas, menos propensas al sobreajuste y mejoran la precisión y estabilidad del modelo.

⚙️ Funcionamiento del Random Forest:

Paso1️⃣- ***Bootstrap Aggregating* (*Bagging*)**

- A partir del conjunto original de 2816 datos con 7612 características (44 × 173) se generan subconjuntos de 2816 datos con todas o no todas (según el hiper parámetro “max_features”) las características. El subconjunto tiene igual número de filas que el conjunto original pero algunas de ellas pueden estar repetidas ya que el muestreo es aleatorio con reemplazo.
- Cada subconjunto se utiliza para entrenar un árbol de decisión independiente. En el presente trabajo unos 200 árboles (según el hiper parámetro “n-estimators”)

Paso 2 - **Construcción de Árboles de Decisión por subconjuntos**

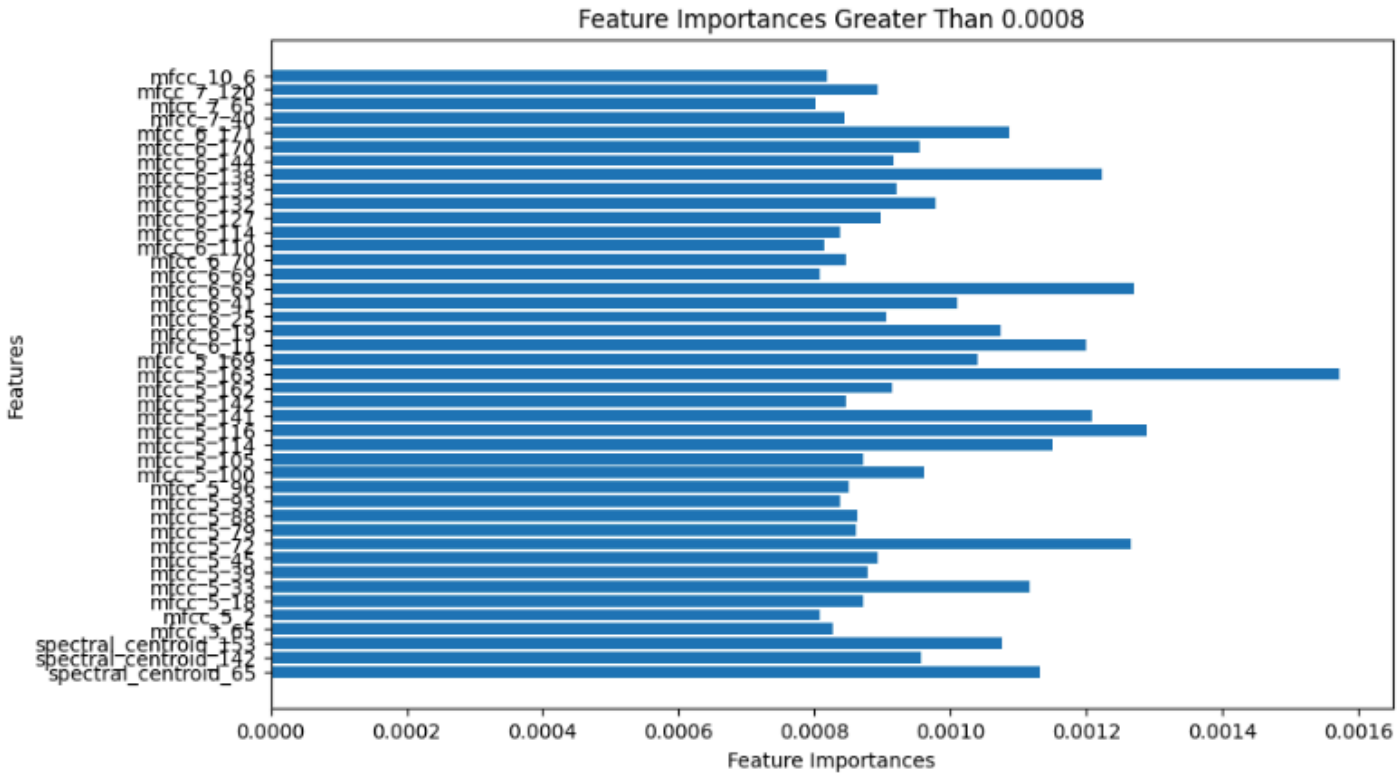
- En vez de construir el árbol de decisión de forma tradicional evaluando todas las características disponibles y eligiendo la mejor, en cada nodo del Random Forest selecciona un subconjunto aleatorio de características y se escoge la mejor característica dentro de ese subconjunto para hacer la división/decisión.

Paso 3 - **Agregación de Predicciones**

- Se toma la predicción de barco que más árboles deciden.

Paso 4 - **Importancia de las características**

Se observan qué características son más importantes que el modelo de Random Forest utiliza para tomar las decisiones de árbol en clasificación de clases.



Lectura y separación de los datos del RF

Para construir lo necesario para el modelo primero se leen y normalizan los datos y después se separan en conjunto de entrenamiento y conjunto de testeo.

Lectura y normalización de los datos

- Se leen los datos generados “features_solas.csv” y se aplica un convertidor para pasar del tipo de dato leído por defecto de *string* a *float*. **code 4**
- Se usa RobustScaler para normalizar los datos para centrar los datos en mediana 0 y escalar la dispersión de los datos con el rango interquartil $z = \frac{x - Me}{IQR}$. Se ha escogido usar este tipo de normalización frente a la típica normalización $z = \frac{x - \mu}{\sigma}$ por qué los datos no siguen una distribución normal. Al usar la mediana y el IQR, los valores atípicos no afectan tanto a la transformación como lo harían con la media y la desviación estándar. **code 5**

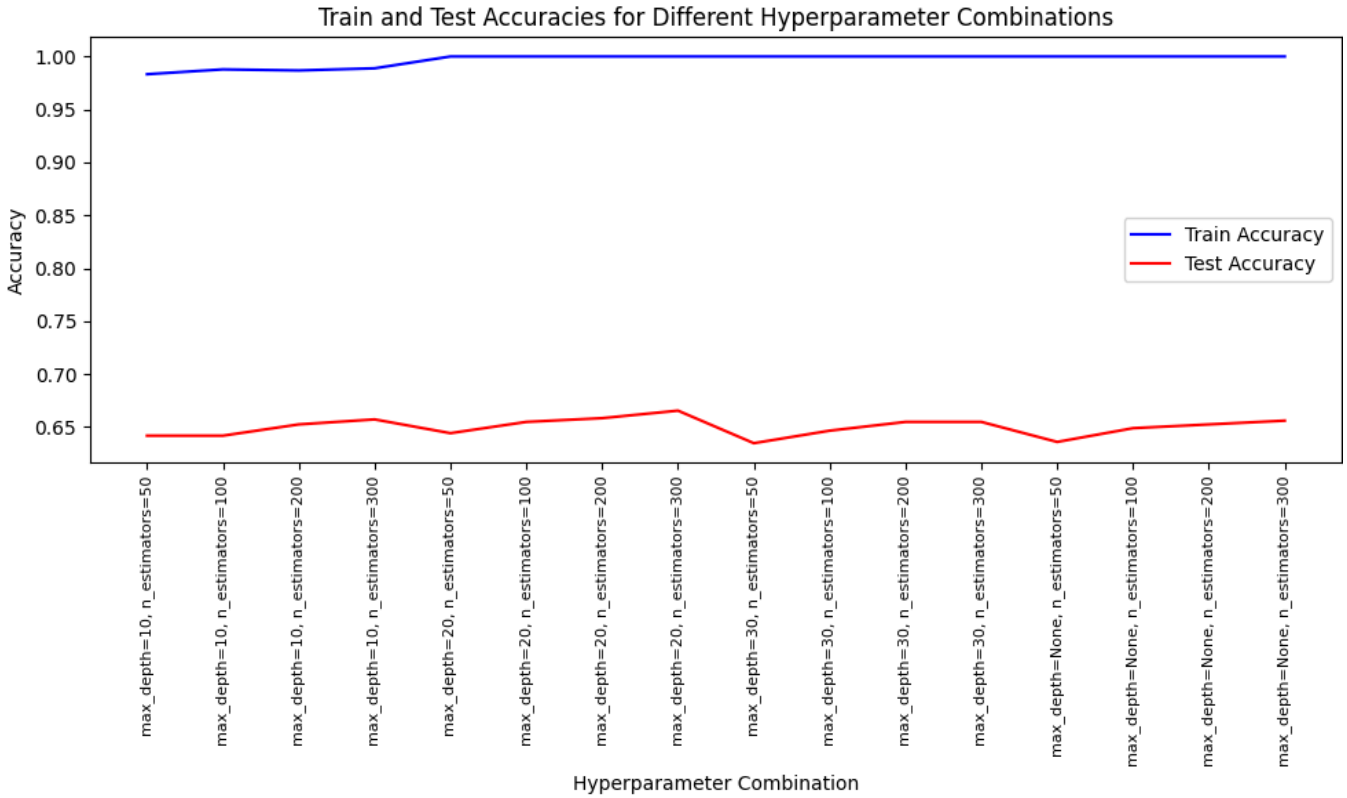
Separación de los datos:

- Se separan los datos con un 30% en test y 70% en entrenamiento. **code 6**

Entrenamiento del RF

El entrenamiento llevado a cabo con Random Forest ha sido con 200 árboles de decisión y de profundidad máxima de 20. **code 7**

Previamente se han hecho pruebas con distintos hiper parámetros para observar posibles mejoras en los resultados. Se han probado 50, 100, 200 y 300 árboles combinados con profundidades máximas de 10, 20, 30 o None. No se han observado mejoras significativas en las precisiones en los datos de entrenamiento o los de testeo. **code 8**


Train and Test Accuracies for Different Hyperparameter Combinations
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Train Accuracy
Test Accuracy
max_depth=10, n_estimators=50
max_depth=10, n_estimators=100
max_depth=10, n_estimators=200
max_depth=10, n_estimators=300
max_depth=20, n_estimators=50
max_depth=20, n_estimators=100
max_depth=20, n_estimators=200
max_depth=20, n_estimators=300
max_depth=30, n_estimators=50
max_depth=30, n_estimators=100
max_depth=30, n_estimators=200
max_depth=30, n_estimators=300
max_depth=None, n_estimators=50
max_depth=None, n_estimators=100
max_depth=None, n_estimators=200
max_depth=None, n_estimators=300
Hyperparameter Combination

Ajustes del desbalance de clases

El desbalance en el conjunto de datos representa un desafío significativo durante el entrenamiento del modelo, especialmente cuando se utilizan algoritmos basados en múltiples árboles de decisión, como Random Forest. Para mitigar este problema, se implementó una estrategia de upsampling mediante SMOTE (Synthetic Minority Over-sampling Technique), la cual genera muestras sintéticas de las clases minoritarias para equilibrar la distribución de datos. Esta técnica permite evitar sesgos hacia las clases mayoritarias y mejora la capacidad del modelo para clasificar de manera equitativa todas las categorías.

KNN (k-Nearest Neighbors)

Como alternativa complementaria, proponemos implementar KNN (K-Nearest Neighbors), considerando que nuestro dataset de entrenamiento es limitado (<3,000 muestras), contexto donde los métodos basados en instancias suelen superar a modelos complejos en eficiencia computacional.

Concepto del k-Nearest Neighbors

El algoritmo k-Nearest Neighbors (k-NN) es un método de aprendizaje supervisado utilizado para clasificación y también para regresión. Su principio fundamental es que una muestra será clasificada según la mayoría de sus **k vecinos más cercanos** en el espacio de características.

⚙️ Funcionamiento del k-Nearest Neighbors en clasificación:

Paso 1️⃣ - **Almacén de datos de entrenamiento**

- Se almacenan los datos de entrenamiento en un espacio n-dimensional de características.

Paso 2️⃣ - **Cálculo de distancias**

- Se calculan las distancias entre el punto a predecir y los puntos de entrenamiento.
- Se utiliza la métrica de distancia por defecto de KNN - Minkowski.

Paso 3️⃣ - **Selección de vecinos**

- Se seleccionan los **k** vecinos más cercanos y se asigna la clase más común entre ellos.
- El valor de k: Un valor bajo puede causar sobreajuste (muy sensible a ruido), mientras que un valor alto puede llevar a un subajuste.
- Ponderación de vecinos: Se puede dar más peso a los vecinos más cercanos en lugar de tratarlos a todos por igual.

Paso 4️⃣ - **Aplicación de un conjunto de datos balanceado**

- Se entrena el mismo modelo con un dataset Upsampling para ver la diferencia.

Lectura y separación de los datos del KNN

Para construir lo necesario para el modelo primero se leen y normalizan los datos y después se separan en conjunto de entrenamiento y conjunto de testeo.

📖 Lectura y normalización de los datos:

- Se leen los datos generados “features_solas.csv” y se aplica un convertidor para pasar del tipo de dato leído por defecto de *string* a *float*. **code 4**
- Se usa *Standard Scaler* para normalizar los datos que es la más recomendada para el KNN

  $z = \frac{x-Min}{Max-Min}$. **code 16**

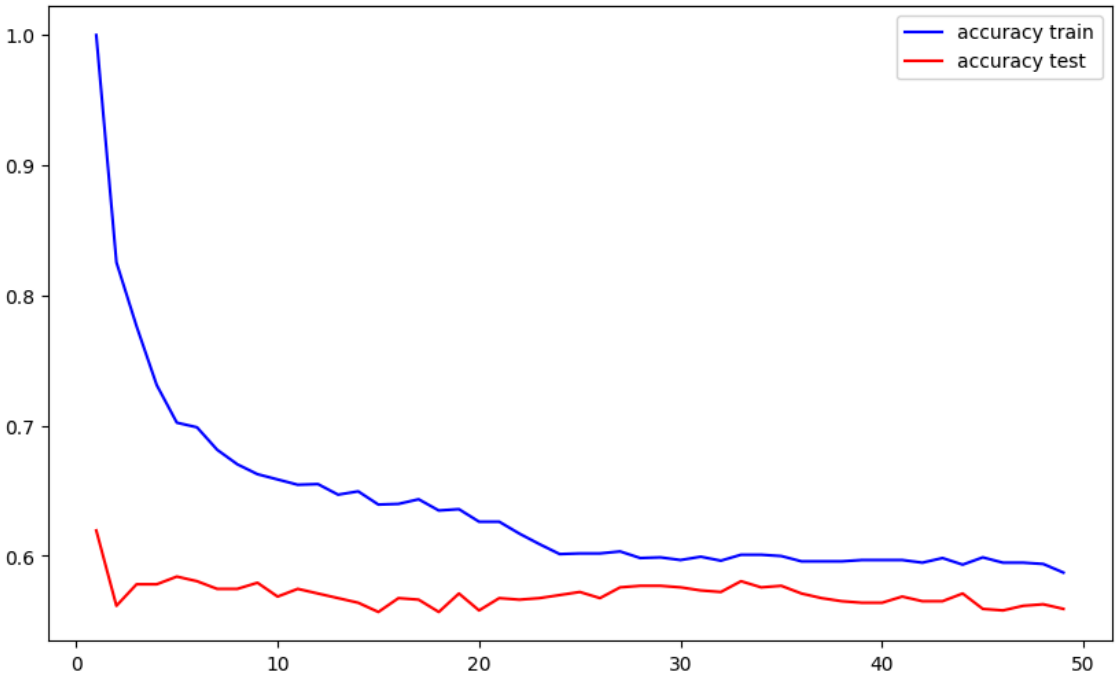
🔀 Separación de los datos: **code 6**

- Se separan los datos con un 30% en test y 70% en entrenamiento.

Entrenamiento del KNN

El proceso de entrenamiento del modelo KNN se inició con una configuración básica utilizando la métrica de Minkowski y un valor de k=1 vecino más cercano. En esta fase inicial, el modelo alcanzó una precisión del 62,88 % en el conjunto de prueba, lo que se podía optimizar.

Para mejorar el rendimiento, se implementó una grid search explorando valores de k en el rango de 1 a 50, combinada con validación cruzada de 5 particiones. Contrario a lo esperado, el análisis reveló que k=1 seguía siendo la opción óptima, lo que sugiere patrones locales altamente definidos en la distribución espacial de los datos. Este comportamiento podría asociarse a una estructura de características donde cada variable (columna) representa un valor correspondiente a un periodo temporal distinto. Dado que la potencia y frecuencia del audio varían en cada periodo y esta diferencia se queda reflejada en los atributos, resulta complejo encontrar un valor de k elevado que prediga correctamente los resultados. Con esa optimización, se logró un aumento de la precisión hasta 66,31 % en el conjunto de pruebas.



Posteriormente, se evaluó el impacto del desbalance de clases utilizando el conjunto de datos preparado en la sección "Balanceo de datos". Tras reentrenar el modelo, se observó una mejora marginal del 2% en la precisión (65,85%), lo que implica que el desbalance inicial no era el factor limitante principal. Este hallazgo sugiere que, aunque k=1 es óptimo en este contexto, podría reflejar sobreajuste.

CNN (Convolutional Neural Networks)

Conceptos del CNN

La selección de redes neuronales convolucionales (CNN) como modelo final para terminar el experimento, derivados de las propiedades intrínsecas de los datos acústicos procesados:

1. Características estructurales de los datos de entrada

Los atributos generados con librosa son matrices 2D (tiempo × frecuencia), similares a imágenes. Las CNN están diseñadas para procesar patrones espaciales en datos estructuralmente organizados, detectando correlaciones locales entre puntos adyacentes en tiempo y frecuencia mediante filtros convolucionales.

2. Ventaja de invarianza traslacional

En audio, patrones clave (ej. una sílaba) pueden aparecer en cualquier posición temporal. Las CNN reconocen estos patrones independientemente de su ubicación gracias al *weight sharing* (mismos filtros aplicados en todas las posiciones), evitando la necesidad de alineación temporal precisa requerida por otros modelos.

3. Aprendizaje jerárquico de características

Las capas convolucionales extraen automáticamente:

- Nivel bajo: Rasgos acústicos locales (ej. formantes, energía en bandas específicas)
- Nivel medio: Combinaciones temporales (ej. transiciones entre fonemas)
- Nivel alto: Semántica abstracta (ej. identificación de instrumentos o palabras clave)

Esto supera la dependencia de características manuales de modelos clásicos como SVM.

Desarrollo de CNN

En este apartado, se describe el desarrollo del tercer modelo basado en una red neuronal convolucional (CNN) para la clasificación de audio en diferentes categorías:

Paso 1 **- Lectura de los datos**

- Se leen los datos generados “features_solas.csv” y se aplica un convertidor para pasar del tipo de dato leído por defecto de *string* a *float*. **code 4**
- El dataset consta de 7612 columnas (44 características X 173 valores temporales) correspondientes a los atributos extraídos y una columna adicional que representa la variable objetivo (*target*).

Paso 2 - **Normalización de datos**

- Dado que el modelo de aprendizaje profundo emplea un algoritmo de optimización basado en el gradiente, la convergencia es más eficiente cuando las características tienen un rango similar. Por ello, como buena práctica, se aplica *StandardScaler* a todas las columnas de características para normalizar los datos.

Paso 3 - **Split de Train/Test**

Una vez normalizado el conjunto de datos, se divide en dos subconjuntos: un 70% de los datos se destina al entrenamiento (*train*) y el 30% restante a la evaluación (*test*).

Paso 4 - **Diseño de la arquitectura de CNN**

La red CNN (AudioCNN en el código) está diseñada para procesar audio a partir de un conjunto de 44 características, donde cada una tiene 173 valores temporales por muestra. Su estructura es la siguiente:

1. **Entrada:** Tensor de dimensión (batch_size, 44, 173) donde:
    - batch_size = 32
    - 44 es el número de características extraídas.

- 173 es la secuencia temporal de cada característica.

Paso 5 - **Capas convolucionales:**

- **Conv1:** 64 filtros de tamaño 3x3, activación *ReLU* y *MaxPooling(2)*
- **Conv2:** 128 filtros de tamaño 3x3, activación *ReLU* y *MaxPooling(2)*

Paso 6 - **Capas completamente conectadas:**

- *Flatten* convierte la salida en un vector de tamaño 128×43.
- *FC1*: 256 neuronas con activación *ReLU*.
- *FC2*: 4 neuronas de salida (Target).

```
AudioCNN(
  (conv1): Conv1d(44, 64, kernel_size=(3,), stride=(1,), padding=(1,))
  (relu): ReLU()
  (pool): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv1d(64, 128, kernel_size=(3,), stride=(1,), padding=(1,))
  (fc1): Linear(in_features=5504, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=4, bias=True)
)
```

code 9 - CNN Diseño 1

Paso 7 - **Definición de la función de optimización y pérdida**

En esta práctica se utiliza el optimizador SDG y CrossEntropyLoss para calcular las pérdidas.

Paso 8 - **Entrenamiento del modelo**

Una vez definidos todos los hiper parámetros y la estructura de la CNN, se procede con la fase de entrenamiento. En la primera prueba, se realizan 20 iteraciones, pero se observa que no es suficiente para una buena convergencia. Por ello, se amplía el entrenamiento hasta 100 iteraciones, donde la reducción de la pérdida se ralentizó considerablemente. Finalmente, en la iteración 100, el modelo alcanzó una pérdida de **0,0015**.

Paso 9️⃣- **Validación del modelo**

Tras finalizar el entrenamiento, se evalúa el rendimiento del modelo en el conjunto de prueba. Los resultados muestran una precisión del 80%, lo que indica un buen desempeño inicial. Sin embargo, se considera que esta precisión aún puede mejorarse significativamente, ya que la arquitectura utilizada es bastante básica.

Paso 🔟 - **Rediseño del CNN**

Para mejorar la precisión del modelo, se rediseña la arquitectura de la CNN incorporando más capas convolucionales, normalización por lotes (*BatchNorm*), *Global Average Pooling* y capas de regularización (*Dropout*).

Descripción de la nueva arquitectura:

1. Capas convolucionales mejoradas:
    - Se agregan tres capas convolucionales con 128, 256 y 512 filtros, todas con activación *ReLU* y normalización por lotes para mejorar la estabilidad del entrenamiento.
2. Global Average Pooling (GAP):
    - En lugar de *MaxPooling*, usamos el *AdaptiveAvgPool1d(1)*, lo que reduce la dimensión temporal y permite que la red extraiga características clave de toda la señal de audio.
3. Capas totalmente conectadas y regularización:
    - En esta estructura se ha añadido tres capas de *"fully connected"* con 512 → 256 → 128 → num_classes (4), usando activaciones *ReLU* y *Dropout (0,5)* para evitar sobreajuste.
4. Optimización:
    - Se cambia el optimizador SDG a *Adam* con una learning rate de 0,001 y la función de pérdida *CrossEntropyLoss* para la clasificación multiclase.

```
AudioCNN(
  (conv1): Conv1d(44, 128, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv1d(128, 256, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv1d(256, 512, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn3): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (global_avg_pool): AdaptiveAvgPool1d(output_size=1)
  (fc1): Linear(in_features=512, out_features=256, bias=True)
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc3): Linear(in_features=128, out_features=4, bias=True)
  (relu): ReLU(inplace=True)
)
```

code 10 - CNN Diseño 2

Con este rediseño de estructura, el modelo alcanza una precisión del **89%**.

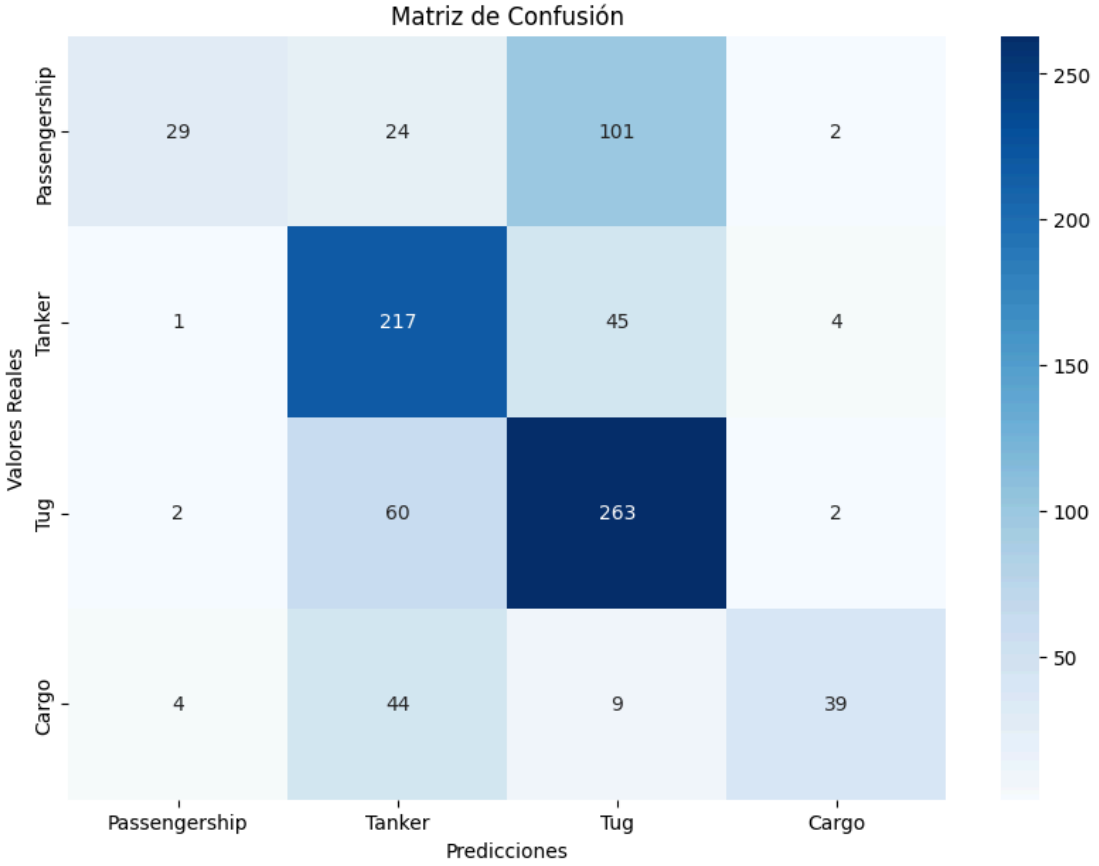
Evaluación de los modelos

RF (Random Forest)

Resultados: Matriz de confusión

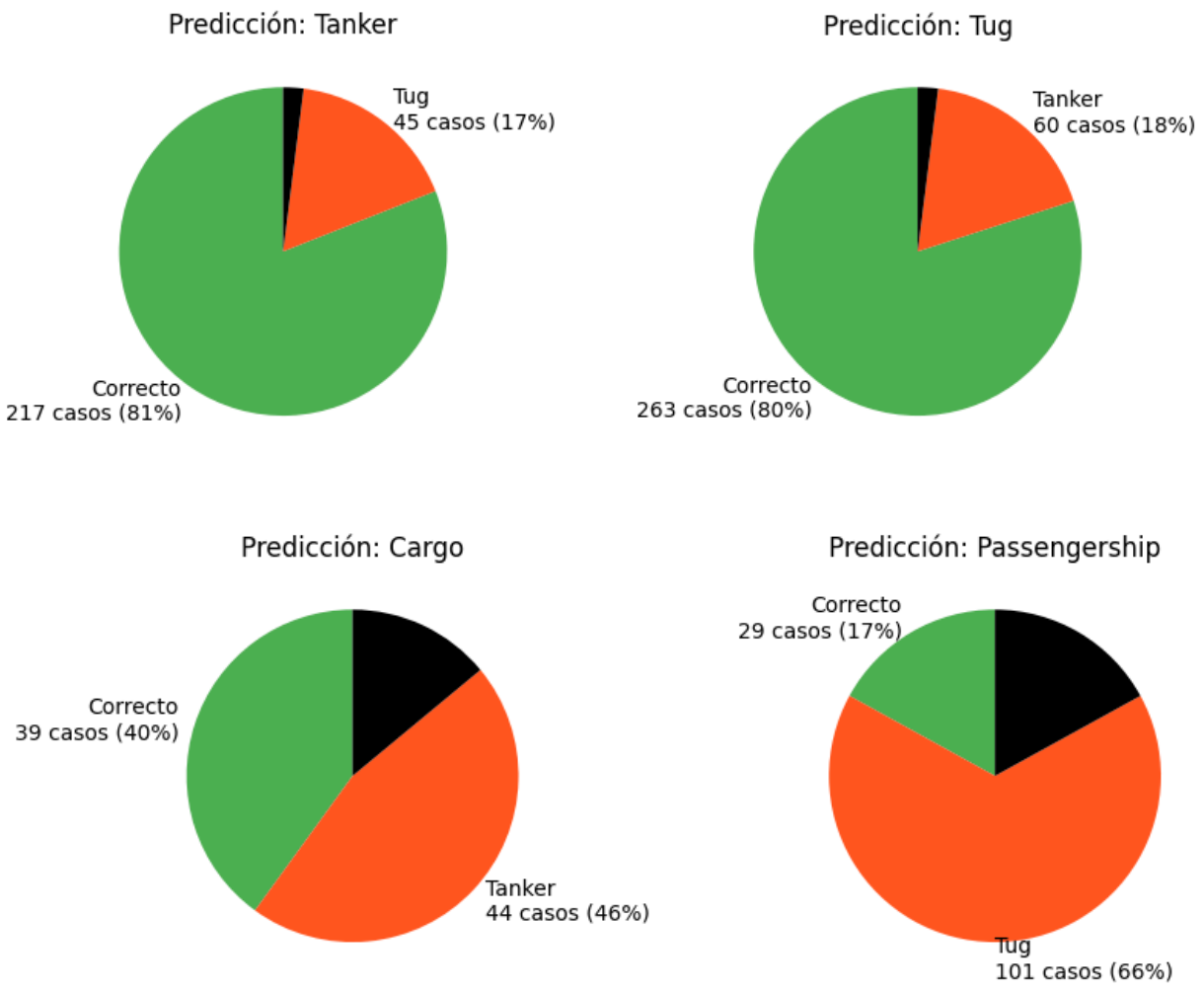
La matriz de confusión compara las predicciones del modelo (y_pred) con los barcos reales de los datos de prueba (y_test). Nos muestra la cantidad de datos que fueron precedidos en cada modo.

code 11



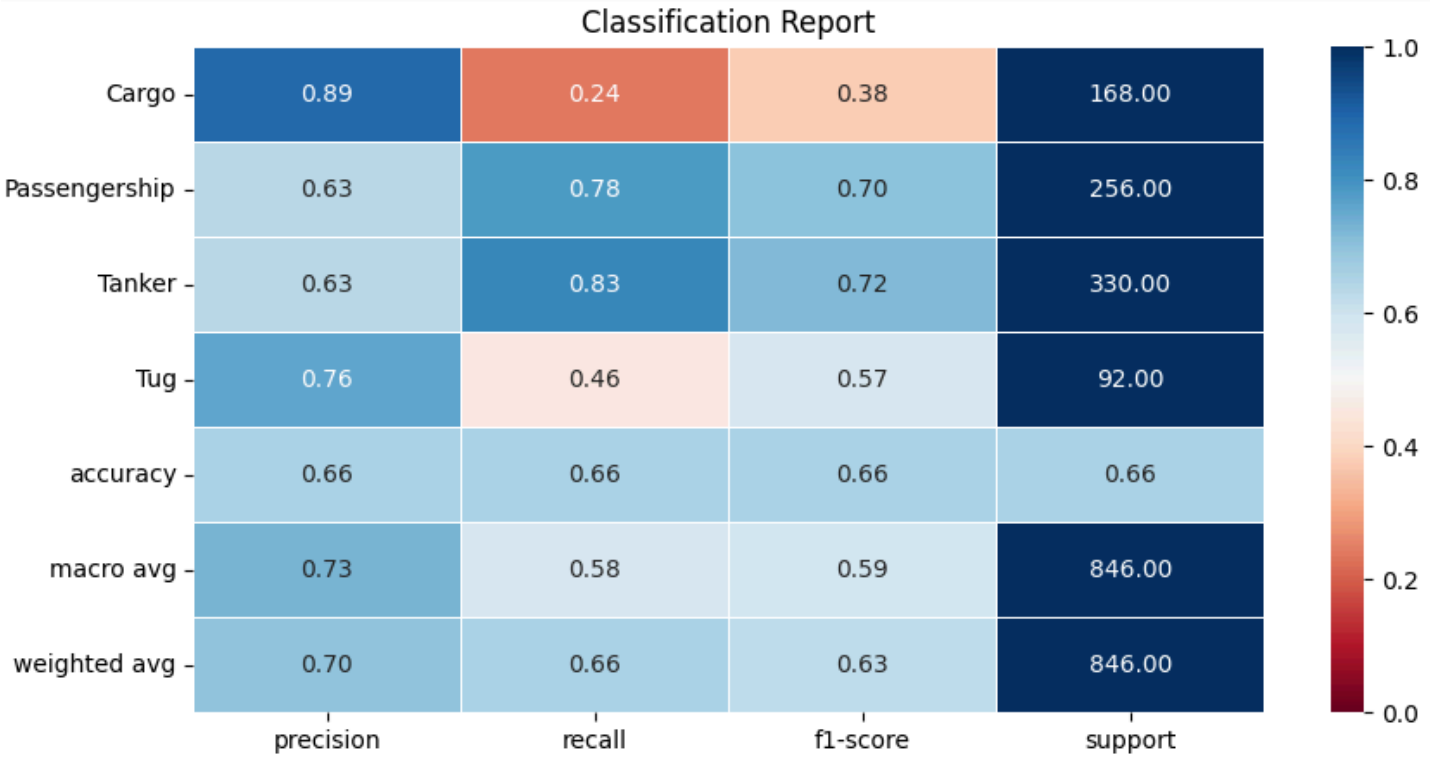
La matriz de confusión del modelo RF permite entender cómo el modelo tiende a sobrepredecir los barcos como Tanker o Tug. Está sesgada a esos dos tipos de barco. Es decir, cuando el modelo falla

es porque identifica el barco como uno de esos tipos. Cuando predice Tanker, acierta en 217 casos (81%) y en los casos que más falla predice que es Tug 45 veces (17%). Cuando predice Tug acierta en 263 veces (80%) y el error más común es confundir por Tanker 60 veces (18%). Cuando predice Cargo solo acierta en 39 casos (40%) fallando confundiendo por Tanker en 44 veces (46%). Cuando predice Passengership acierta en sólo 29 casos (17%) y confunde por Tug en 101 ocasiones (66%). En resumen, confunde Cargos por Tankers y Passengerships por Tugs. El siguiente diagrama de pastel resume esta información de forma más visual. **code 12**



Resultados: Precisión, Recall y F1-Score

Se verifica el modelo RF con las métricas de Precisión, Recall y F1-Score. **code 13**

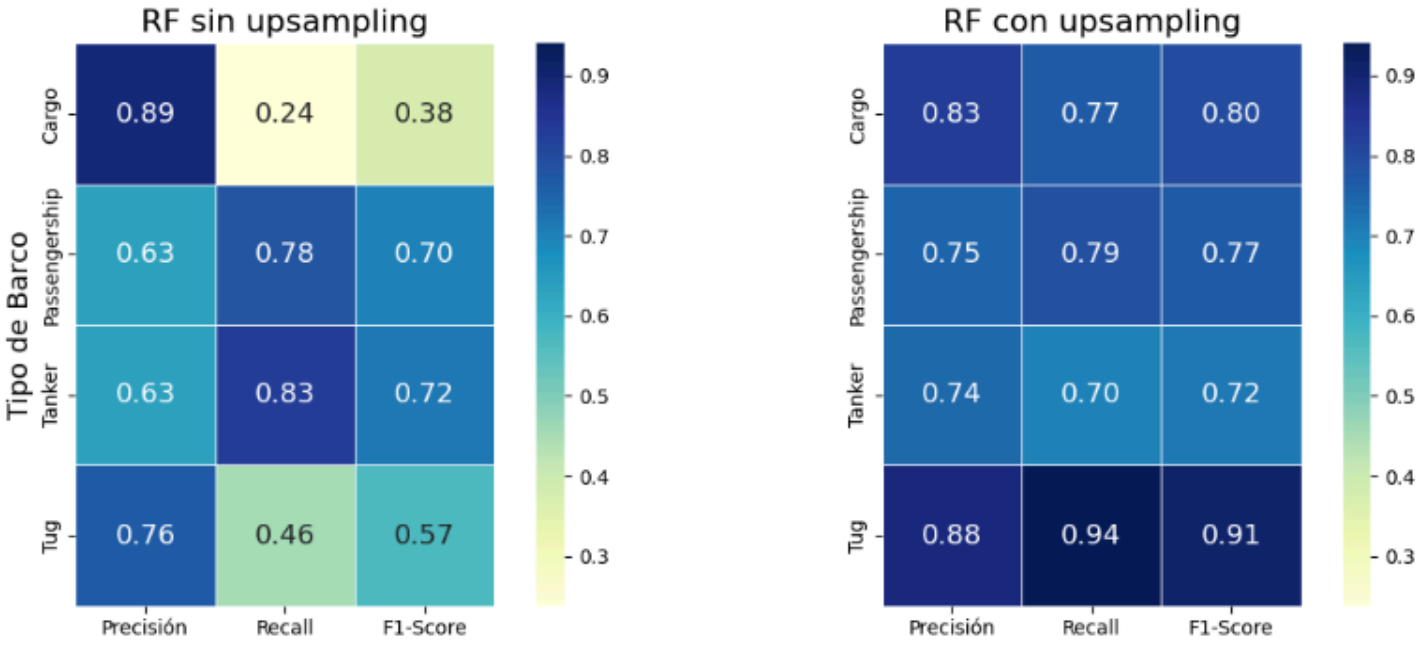


- Por lo que respecta a la **precisión**, el modelo ofrece probabilidades superiores al 63% de que la predicción sea correcta. Cargo es la clase que ofrece menor cantidad de falsos positivos teniendo la mayor precisión del 89%. Si el modelo indica que es Cargo, estamos bastante seguros que será Cargo. En cambio por las otras clases es algo más probable (37% o 24%) que sea otra clase mal indicada.
- En los barcos Passengership y Tanker se observa una **recuperación** más alta (0,78 y 0,83) que en los barcos Cargo y Tug (0,24 y 0,46) que es baja. Esto indica que el modelo es capaz de detectar los casos que son positivos en Passengership y Tankers, que son las clases que más datos tuvo el modelo para detectar. La recuperación baja en Cargo y Tug indica que pocos casos que eran positivos fueron detectados correctamente por el modelo. Es decir, que se pierden bastantes instancias positivas.

- En los barcos Passengership y Tanker se observa una **f1-score** más alta (0,70 y 0,72) que en los barcos Cargo y Tug (0,38 y 0,57) que es baja. El f1-score bajo en Cargo y Tug indica que la precisión como la recuperación están desequilibradas. El f1-score alto en Passengership y Tanker indica que la precisión como la recuperación están equilibradas. Esto es debido a un desequilibrio entre las clases ya que Cargo y Tug tienen menor cantidad de datos comparado con las otras dos.
- En **support** se observa que este es proporcional a los datos con los que se contaba en el conjunto de datos, hecho que lo hace coherente con la estratificación realizada en el train test split. También es notorio la relación entre el recall bajo con un support bajo indicando que se recuperan correctamente menos casos en los barcos que se tenía menos información (menos datos) y en cambio se recuperan mejores los barcos de los cuales se tenían mayores datos. El modelo parece estar mejor entrenado para recuperar los tipos que tenía mayor información.

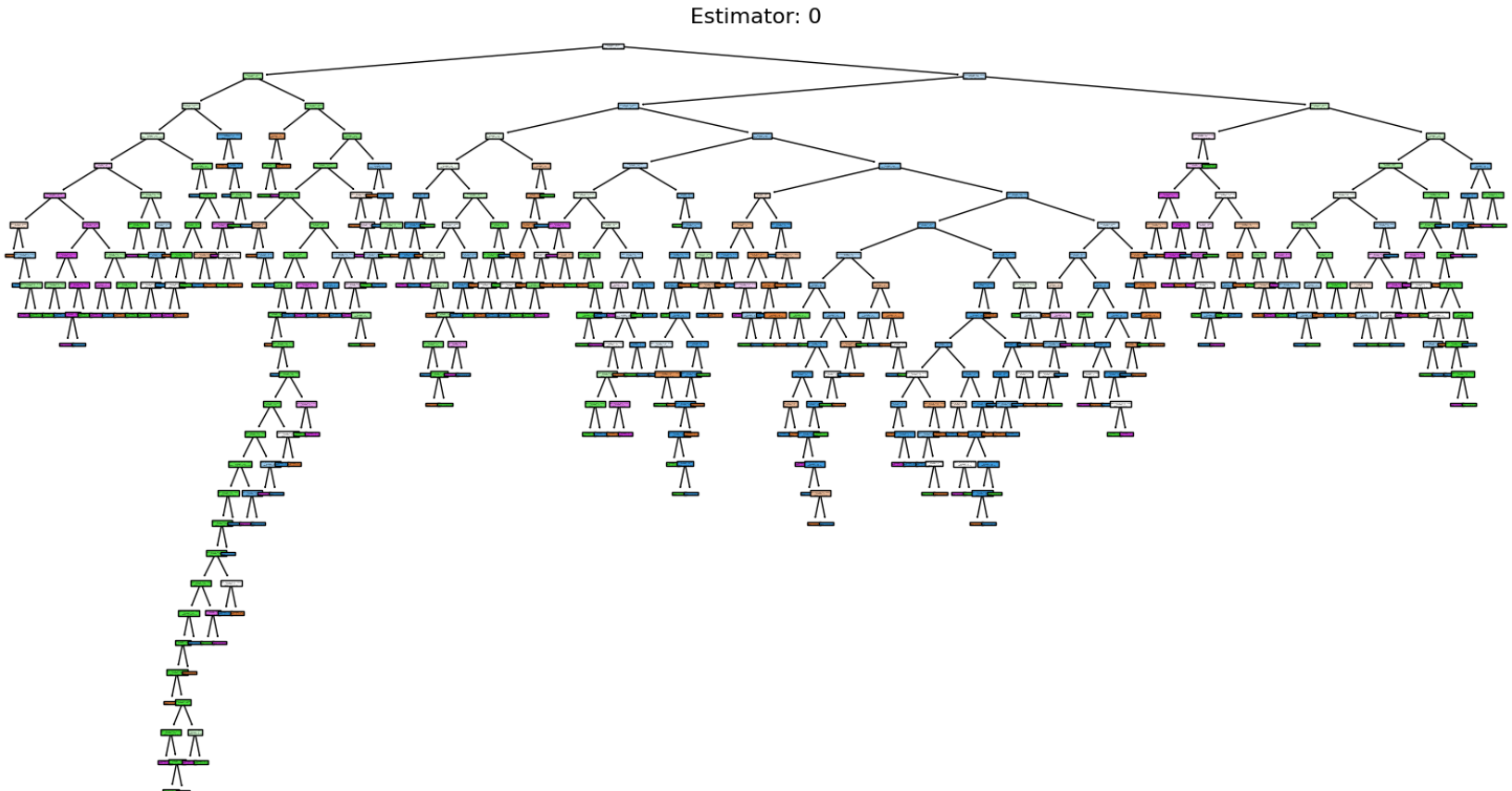
Ajustes del desbalance de clases

Como resultado, al entrenar el modelo Random Forest con el dataset ajustado mediante upsampling, se observó una mejora notable en el rendimiento, alcanzando una precisión del **80%**.

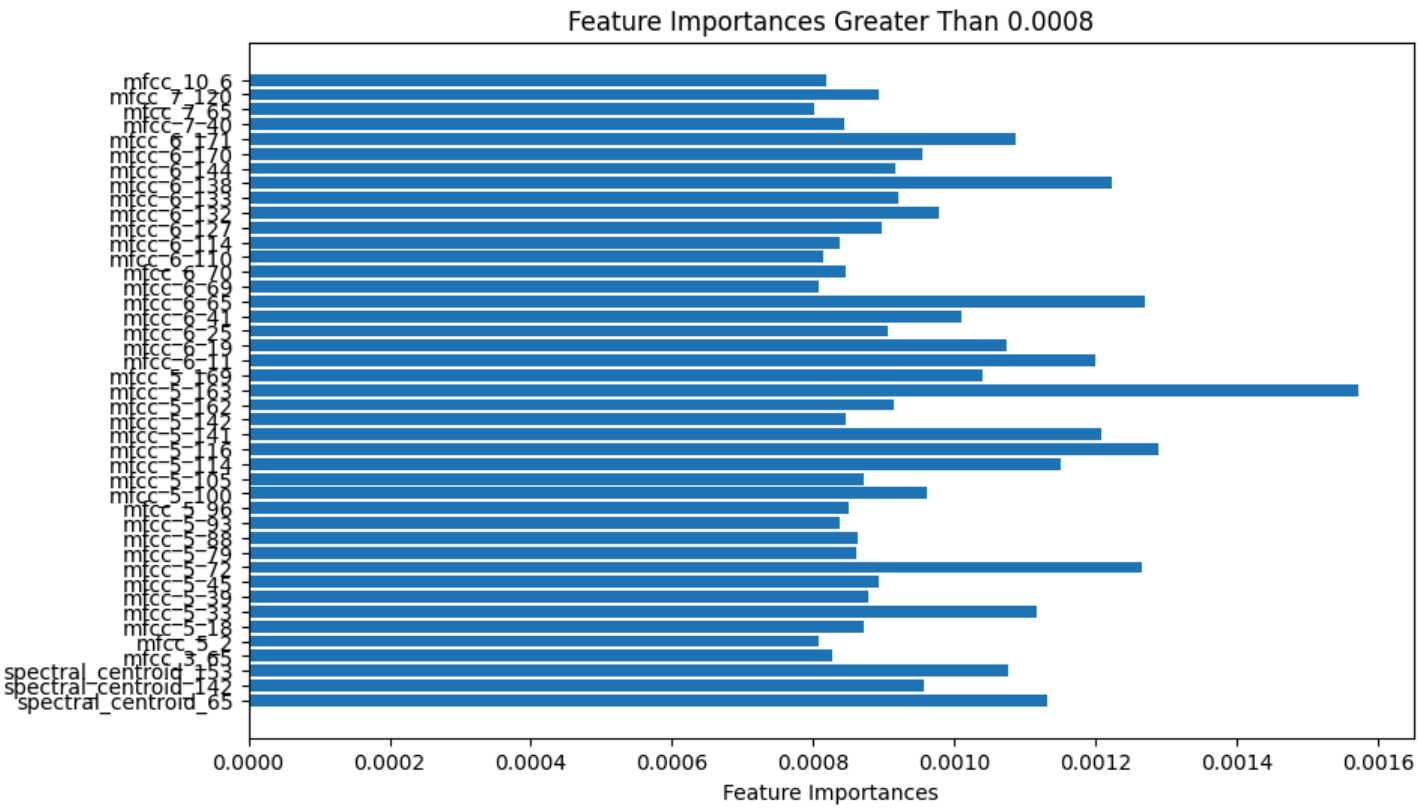


Resultados: Gráfica de árboles

Un árbol de decisión nos ilustra la explicabilidad del modelo en la clasificación. Cada nodo del árbol representa una característica y cada borde conecta nodos representando las decisiones tomadas basadas en esa característica (eg: si la característica es mayor qué tal valor, entonces es ese barco). Las hojas del árbol representan las clases a las que pertenecen los datos. Los gráficos en árbol ayudan a entender cómo cada árbol ha tomado decisiones para clasificar y proporciona información sobre la complejidad del modelo y la importancia relativa de las características en la toma de decisiones del clasificador. Sin embargo, como se destacó en las desventajas en Random Forest, esta explicabilidad se pierde al usar múltiples árboles de decisión y usar sus votaciones. **code 14**



Resultados: Características importantes

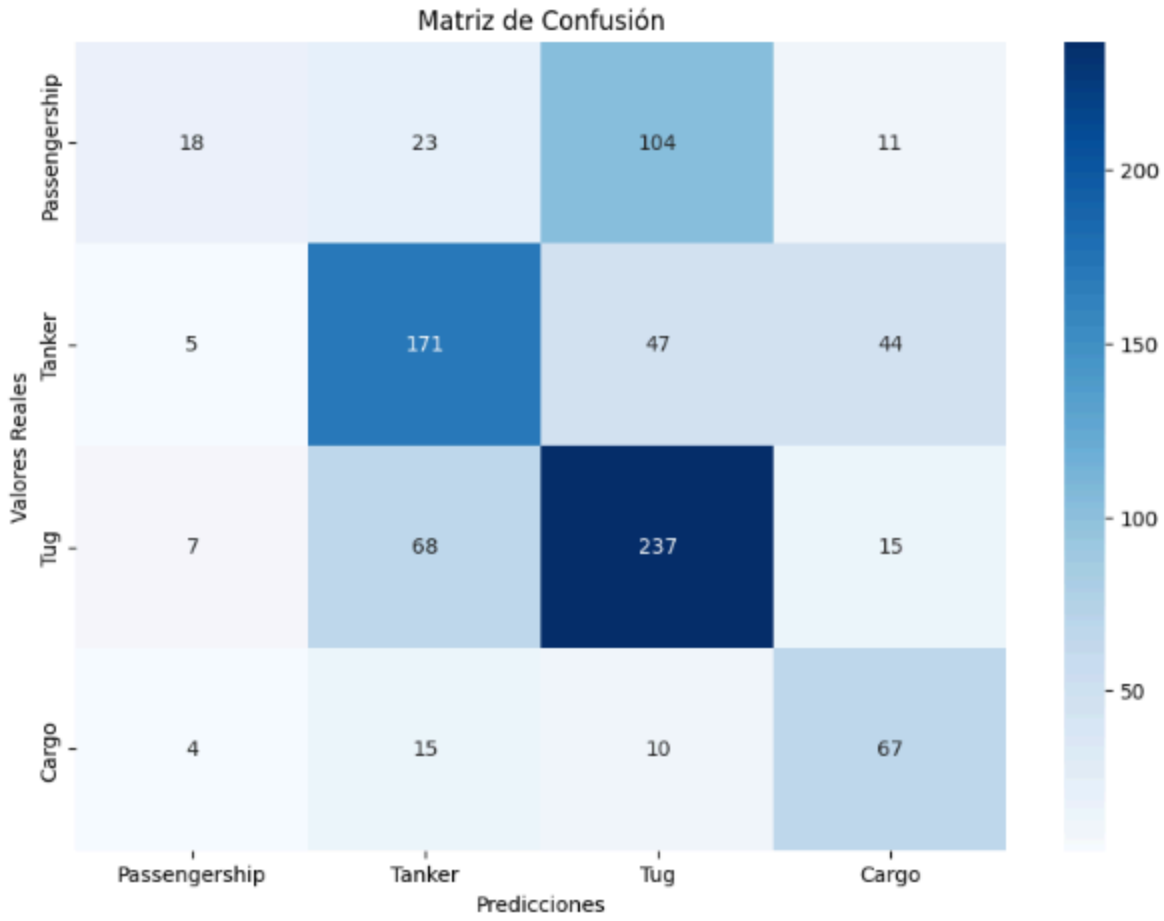


code 15

KNN (k-Nearest Neighbors)

Resultados: Matriz de confusión

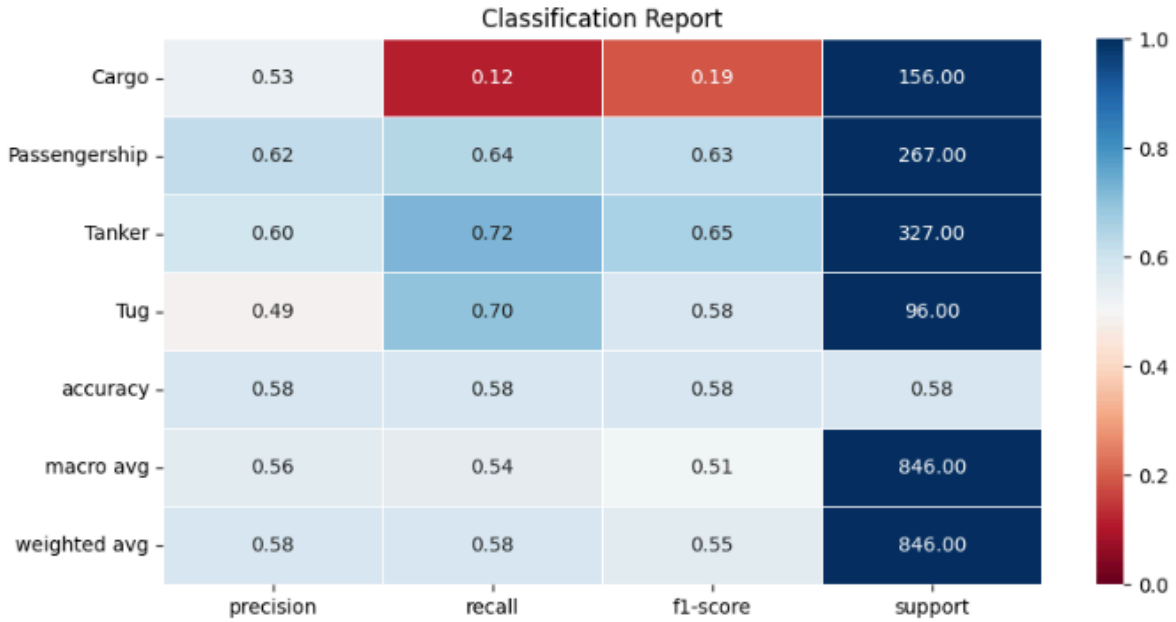
La matriz de confusión compara las predicciones del modelo (y_pred) con los barcos reales de los datos de prueba (y_test). Nos muestra la cantidad de datos que fueron precedidos en cada modo.



Los resultados de la matriz de confusión sugieren que el modelo predice de forma incorrecta la classe Passengerhip, ya que la confunde muy frecuentemente por la clase Tug. Además, se observa que en muchas pocas ocasiones el modelo predice Passengership (15+5+7+4=31). La clase Tanker a veces es confundida por Tug o Cargo aunque acierta en un 64% (171/(5+171+47+44)) de los casos. La clase Tug es la que el modelo

mejor predice aunque confunde por Tanker en un 20% de las ocasiones, juntamente con la clase Cargo que relativamente también es bien predecida por el modelo en un 70% de los casos.

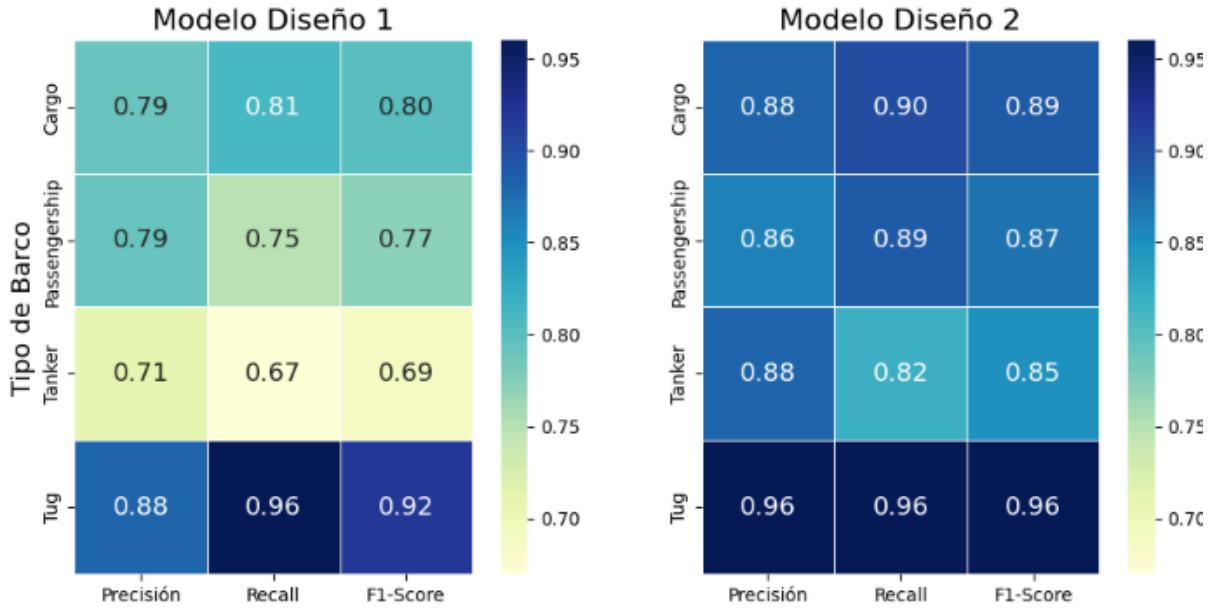
Resultados: Precisión, Recall y F1-Score



- Las **precisiones** del modelo a lo largo de las clases no son muy altas. Las predicciones en las clases Passengership y Tug son un 60% probabilidad de que sean correctas mientras que de la clase Cargo y Tug se llega a que solo se tiene la certeza de un 50% que la predicción sea correcta.
- Como iba indicando ya la matriz de confusión y ahora nos indica el muy bajo **recall**, pocas clases Cargo fueron detectadas correctamente por el modelo. Sin embargo, para las clases Tanker y Tug el modelo es capaz de detectar los casos positivos y se pierde pocas instancias positivas de esas clases.
- La **f1-score** alrededor de 0,60 indican que el recall y precisión están regularmente equilibradas, es decir el modelo tiene algunos falsos negativos (precisión baja) y falsos positivos (recall). En el caso de cargo, la f1-score es baja debido a su ya bajísima recall.
- En **support** se observa que este es proporcional a los datos con los que se contaba en el conjunto de datos, hecho que lo hace coherente con la estratificación realizada en el train test split.

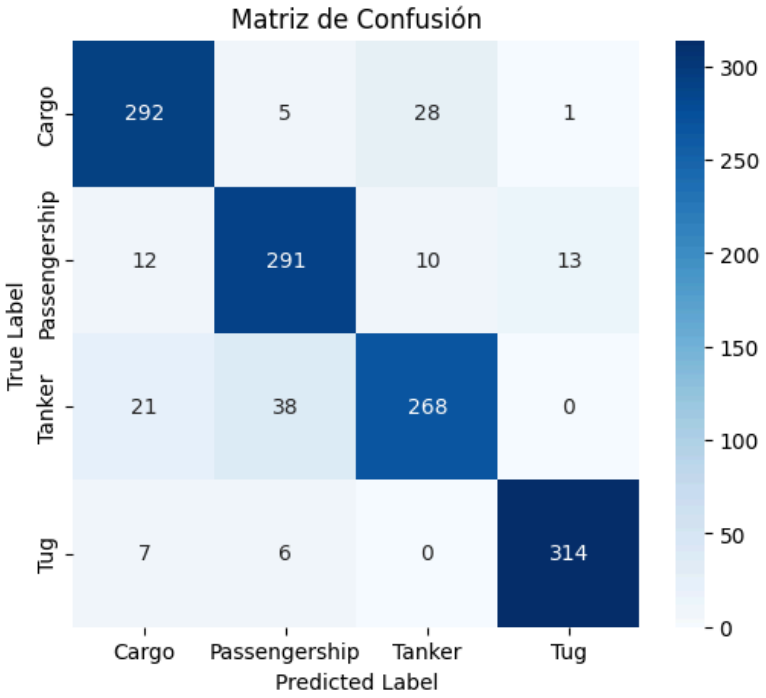
CNN (Convolutional Neural Networks)

Resultados: Precisión, Recall y F1-Score



- **Precisión general**: La mejora de la arquitectura con capas adicionales permitió un incremento significativo en la precisión general del modelo, donde alcanza hasta un promedio de 89% de accuracy.

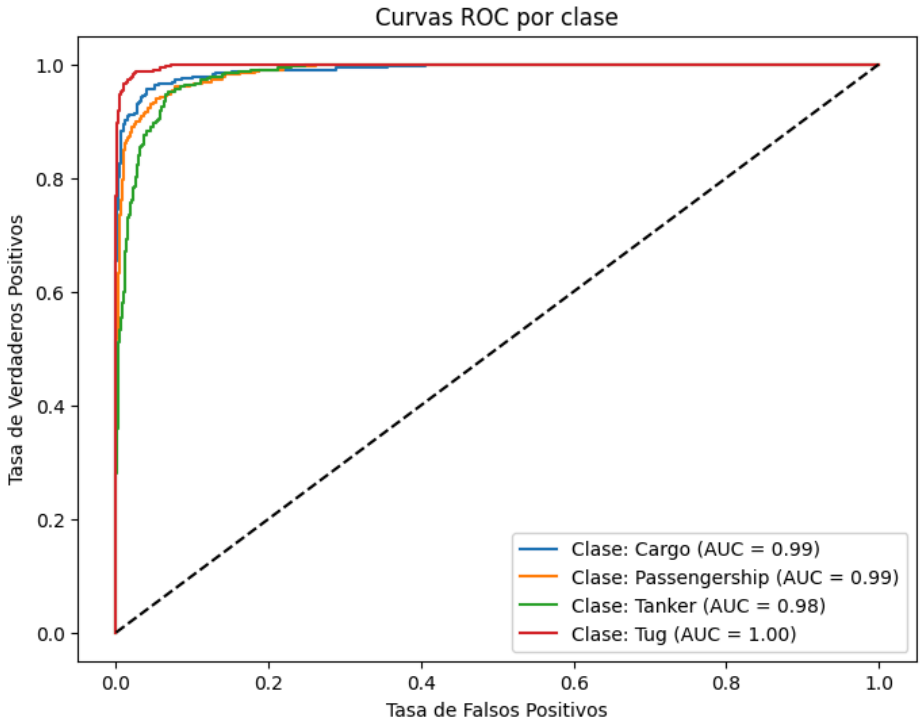
Resultados: Matriz de confusión



Se observa que la mayoría de las clases son correctamente clasificadas, con mínimos errores entre "Tanker" y "Passengership".

Resultados: Curvas ROC

Se visualizaron las curvas ROC para cada clase, mostrando una área bajo la curva (AUC) cercana a 1, lo que indica un buen rendimiento del modelo.



Conclusiones

El presente trabajo ha permitido desarrollar un sistema de clasificación de audios submarinos para evaluar el impacto del ruido marítimo en especies marinas, implementando un pipeline de minería de datos con tres modelos de machine learning. A continuación, se resumen las conclusiones clave:

Impacto del preprocesamiento:

- La normalización con RobustScaler/StandardScaler y el balanceo mediante SMOTE fueron críticos para mejorar la generalización, especialmente en RF y CNN.
- La elección de parámetros (frame size=4096, hop length=512) optimizó la resolución en frecuencias bajas (<1000 Hz), clave para distinguir sonidos de barcos.

Comparación del rendimiento de los modelos:

- **Random Forest (RF)**: Alcanzó un 80% de precisión tras balancear los datos con un Upsampling, demostrando robustez en datos tabulares y capacidad para identificar características relevantes. Sin embargo, mostró sesgo hacia clases mayoritarias (Tanker y Tug), con bajo recall en clases minoritarias (Cargo: 24%, Passengership: 17%).
- **KNN**: Se obtuvieron precisiones del 66,31% o del 65,85% con la introducción de datos sintéticos para balancear los datos. Estos resultados evidencian las limitaciones del modelo KNN ante la alta dimensionalidad de características y los datos desbalanceados. Su dependencia de patrones locales y sensibilidad al ruido lo hacen menos adecuado para este contexto.
- **CNN**: Destacó con un 89% de precisión tras optimizar su arquitectura (capas convolucionales profundas, BatchNorm y Dropout), validando su buena capacidad para capturar patrones espacio-temporales en espectrogramas y características acústicas.

Limitaciones y desafíos:

- El desbalance de clases afectó significativamente a RF y KNN, subrayando la necesidad de estrategias adicionales como por ejemplo incluir la recolección de más datos o realizar un *cost-sensitive learning* (técnica que asigna un mayor peso o penalización a los errores cometidos en las clases minoritarias).

- La alta dimensionalidad de las características (44 × 173) incrementó la complejidad computacional, especialmente en KNN.

Líneas futuras de trabajo:

- Optimizar la selección de características: Seleccionar las características prioritarias que proporcionan mayor *feature importance* para reducir redundancia y mejorar eficiencia, sobre todo en los modelos RF y KNN.
- Explorar más opciones de validación cruzada en KNN ya que en el presente trabajo solo se usaron 5 folds.
- Implementar CNN con atención temporal (*Transformers*) que permite al modelo enfocarse en distintas partes clave de la secuencia al mismo tiempo sin necesidad de procesarlas en iteraciones por ventanas.
- Implementar modelos híbridos CNN + LSTM (*Long Short-Term Memory*) ya que ayuda a modelar las dependencias temporales de secuencias largas con una arquitectura que está basada en puertas que controlan qué información se mantiene o se olvida en cada paso de la secuencia temporal.
- Ampliar el conjunto de datos con grabaciones en escenarios realistas (múltiples barcos, ruido ambiental) teniendo en cuenta que estas grabaciones estén balanceadas.
- Usar el dataset sin las fragmentaciones de los audios.

En síntesis, este trabajo valida el potencial de las CNN para clasificación acústica submarina, ofreciendo una solución escalable y precisa para el monitoreo automatizado de ruido marítimo, con implicaciones directas en la conservación de ecosistemas marinos.

Anexo

code 0 - Formas y escuchar audio

Se prueba con distintos audios [5] [23], etc

```
y, sr = lb.load(df["audio_path"][5])
# Mostrar la forma de onda del audio
plt.figure(figsize=(14, 5))
lb.display.waveshow(y, sr=sr)
plt.title('Forma de Onda del Audio 5')
plt.show()
# Escuchar el audio
ipd.Audio(y, rate=sr)
```

code 1 - Generación Espectrogramas

El mismo código se hace para los 4 tipos de barco: Cargo, Tug, Tanker y Passengership. Ejemplo para el barco Cargo:

```
cargo = df[df["ship"] == "Cargo"] # Se filtran solo los barcos Cargo
cargo["audio_path"][2297] # Ejemplo de un path

import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import librosa
# Cargar un archivo
y, sr = lb.load(cargo["audio_path"][2297])
# Configurar los parámetros del espectrograma
frame_size = 4096  # Tamaño de la ventana de análisis en muestras
hop_length = frame_size // 8  # Tamaño de la ventana de análisis en muestras
  # Desplazamiento entre ventanas en muestras
```

```
# Calcular el espectrograma usando la transformada de Fourier de
tiempo corto (STFT)
D = np.abs(librosa.stft(y, n_fft=frame_size, hop_length=hop_length))
# Convertir a decibeles (escala logarítmica) con potencia 2
DB = librosa.amplitude_to_db(D, ref=np.max)
# Mostrar el espectrograma usando matplotlib
plt.figure(figsize=(10, 6))
librosa.display.specshow(DB, sr=sr, hop_length=hop_length, x_axis='time',
y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title('Espectrograma de un cargo')
plt.xlabel('Tiempo')
plt.ylabel('Frecuencia')
plt.tight_layout()
plt.show()
```

code 2 - Comparación ZCR con distintos tamaños del frame

```
# Cargar audio
y, sr = librosa.load(tanker["audio_path"][320], sr=None)
y = y - (min(y) + (max(y) - min(y)) / 2)  # Para situar el 0 en el medio del mínimo
y el máximo.
# Definir diferentes tamaños de ventana
frame_sizes = [4096//2, 4096, 4096*2]

plt.figure(figsize=(10, 5))
for frame_size in frame_sizes:
    hop_length = frame_size// 8
    zcr = librosa.feature.zero_crossing_rate(y, frame_length=frame_size,
hop_length=hop_length)
    plt.plot(zcr[0], label=f"Frame: {frame_size}")
plt.xlabel("Frames")
plt.ylabel("ZCR")
```

```
plt.title("Comparación de ZCR con diferentes tamaños de frame")
plt.legend()
plt.show()
```

code 3 - Valores Hop_Length y Frame_Size usados

```
print("Muestras por segundo = ", sr, "frames")
print("Muestras por audio = ", 4*sr, "frames\n")

FRAME_SIZE = 4096 # Muestras que tiene la ventana de audio analizada.
HOP_LENGTH = FRAME_SIZE // 8 # Muestras que se saltan de ventana en ventana.

print("HOP_LENGTH = ", HOP_LENGTH, "frames")
print("HOP_LENGTH = ", round(HOP_LENGTH/sr,3), "segundos\n")
print("FRAME_SIZE = ", FRAME_SIZE, "frames")
print("FRAME_SIZE = ", round(FRAME_SIZE/sr,3), "segundos")
```

demostración 4

Hay distintas formas de demostrar que energía, potencia e intensidad son proporcionales a la amplitud al cuadrado:

Forma 1

La energía del movimiento armónico simple es:

$$E=\frac{1}{2}kA^2$$

$$k=m\omega^2$$

$$\omega=\frac{2\pi}{T}=2\pi f$$

Energía de una onda

$$E=2\pi^2 f^2 A^2 m$$

La potencia es la energía por unidad de tiempo.

$$P=\frac{dE}{dt}=2\pi^2 f^2 A^2 \frac{dm}{dt}$$

La intensidad es la potencia por unidad de espacio que define el frente de onda.

- Unidimensional: $I = P$ $\quad P = 2\pi^2 f^2 A^2 \frac{dm}{dt}$
- Bidimensional: $I = \frac{P}{L}$
- Tridimensional: $I = \frac{P}{S}$ $\quad \boldsymbol{I = 2\pi^2 f^2 \rho v A^2}$

Forma 2:

- Energía potencial elástica de un movimiento armónico simple: $E = \frac{1}{2} k A^2$
- Fuerza elástica de tensión: $T = k L$ Donde k = constante elástica, L = longitud del resorte
- Tensión de una cuerda tensa: $T = \mu \cdot v^2$ Donde v = velocidad de propagación, y μ la densidad lineal del medio: $\mu = \frac{m}{\Delta x}$

Con todo, sustituyendo expresiones de arriba abajo: $E = \frac{1}{2} k A^2 = \frac{1}{2} \frac{T}{L} A^2 = \frac{1}{2} \frac{\mu \cdot v^2}{L} A^2$

Para la potencia (energía por unidad de tiempo): $P = \frac{E}{t} = \frac{1}{2} \frac{\mu \cdot v^2}{L \cdot t} A^2$

Para la intensidad (potencia por unidad de espacio del frente de onda, en este caso tridimensional):

$$I = \frac{P}{4\pi r^2} = \frac{1}{8\pi r^2} \cdot \frac{\mu \cdot v^2}{L \cdot t} A^2$$

code 4 - Lectura de los datos en los modelos RF, KNN, CNN

```
def to_float_list(value):
    # Se eliminan los corchetes "[" y "]" del principio y el final → usando
strip("[]")
    # Se separan los números por el espacio en blanco →  usando split()
    # Se eliminan las comas sobrantes →  usando strip(',')
    # Se aplica la función float a cada elemento de la lista → usando map()
    return float(value.strip("[]").split()[0].strip(','))

# Se lee el CSV
df = pd.read_csv(output_folder + "/features_solas.csv")

# Las columnas "ship", "date" y "audio_path" no se procesan por la función
to_float_list ya que no son valores float
converters = {col: to_float_list for col in df.columns if col not in ["ship",
"date", "audio_path"]}

# Se lee el CSV aplicando los conversores
df = pd.read_csv(output_folder + "/features_solas.csv", converters=converters)
df.tail()
```

code 5 - RF y KNN normalización con RobustScaler

```
 ## Normalizar los datos con RobustScaler
# Seleccionar solo las columnas numéricas
numerical_cols = df.select_dtypes(include=['number']).columns

# Inicializar y aplicar RobustScaler
scaler = RobustScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

code 6 - RF train test split

```
## Separación el dataframe según las características "X" y el target "y".
X = df.drop(columns=["date", "audio_path", "ship"]) # Se quita el target y las
```

```
caracerísticas que no se tienen que analizar

# Convertir los datos de X de string a float

# El target es adivinar el barco.
y = df["ship"]

# División estratificada para mantener la proporción de clases en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=23, stratify=y)

# Verificar la proporción de clases en train y test
print("Proporción en train:")
print(y_train.value_counts(normalize=True))

print("Proporción en test:")
print(y_test.value_counts(normalize=True))
```

code 7 - RF entrenamiento

```
X_train = X_train.values
y_train = y_train.values

RF = RandomForestClassifier(max_depth=20, n_estimators = 200, random_state=0)
RF.fit(X_train, y_train)
y_train_pred = RF.predict(X_train)
y_test_pred = RF.predict(X_test)
```

code 8 - RF entrenamiento con distintos hiper parámetros

```
param_grid = {
    'max_depth': [10, 20, 30, None],
    'n_estimators': [50, 100, 200, 300]}
```

```
# Crear listas para almacenar las accuracies
train_accuracies = []
test_accuracies = []
param_combinations = []  # Lista para almacenar las combinaciones de parámetros

# Dividir los datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=23)

# Realizar un bucle sobre todas las combinaciones de los parámetros
for max_depth in param_grid['max_depth']:
    for n_estimators in param_grid['n_estimators']:
        # Crear el modelo con los parámetros actuales
        RF = RandomForestClassifier(
            max_depth=max_depth,
            n_estimators=n_estimators,
            random_state=23
        )

        # Entrenar el modelo
        RF.fit(X_train, y_train)

        # Predecir
        y_train_pred = RF.predict(X_train)
        y_test_pred = RF.predict(X_test)

        # Calcular las accuracies
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)

        # Almacenar las accuracies
        train_accuracies.append(train_accuracy)
        test_accuracies.append(test_accuracy)
```

```
        # Almacenar la combinación de parámetros
        param_combinations.append(f"max_depth={max_depth},
n_estimators={n_estimators}")

# Graficar los resultados
plt.figure(figsize=(10, 6))
plt.plot(range(len(train_accuracies)), train_accuracies, label="Train Accuracy",
color='blue')
plt.plot(range(len(test_accuracies)), test_accuracies, label="Test Accuracy",
color='red')
plt.xticks(ticks=range(len(param_combinations)), labels=param_combinations,
rotation=90, fontsize=8)
plt.xlabel("Hyperparameter Combination")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Train and Test Accuracies for Different Hyperparameter Combinations")
plt.tight_layout()  # Ajustar el diseño para evitar solapamiento de etiquetas
plt.show()
```

code 9 - CNN Diseño 1

```
class AudioCNN(nn.Module):
    def __init__(self, n_features, num_classes):
        super(AudioCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=n_features, out_channels=64,
kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool1d(kernel_size=2)  # Reduce time_steps a la mitad
        self.conv2 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3,
padding=1)
        self.fc1 = nn.Linear(128 * 43, 256)  # Ajustado a time_steps=173
        self.fc2 = nn.Linear(256, num_classes)
```

```
    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))  # (batch, 64, 86)
        x = self.pool(self.relu(self.conv2(x)))  # (batch, 128, 43)
        x = torch.flatten(x, start_dim=1)       # (batch, 128*43)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

code 10 - CNN Diseño 2

```
class AudioCNN(nn.Module):
    def __init__(self, n_features, num_classes):
        super(AudioCNN, self).__init__()

        # Capas convolucionales
        self.conv1 = nn.Conv1d(in_channels=n_features, out_channels=128,
kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm1d(128)
        self.conv2 = nn.Conv1d(in_channels=128, out_channels=256, kernel_size=3,
padding=1)
        self.bn2 = nn.BatchNorm1d(256)
        self.conv3 = nn.Conv1d(in_channels=256, out_channels=512, kernel_size=3,
padding=1)
        self.bn3 = nn.BatchNorm1d(512)

        # Global Average Pooling
        self.global_avg_pool = nn.AdaptiveAvgPool1d(1)

        # Capas fully connected
        self.fc1 = nn.Linear(512, 256)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, 128)
```

```
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(128, num_classes)

        # Funciones de activación
        self.relu = nn.ReLU(0.1)

    def forward(self, x):
        # Capas convolucionales
        x = self.relu(self.bn1(self.conv1(x)))  # (batch, 128, 173)
        x = self.relu(self.bn2(self.conv2(x)))  # (batch, 256, 173)
        x = self.relu(self.bn3(self.conv3(x)))  # (batch, 512, 173)

        # Global Average Pooling
        x = self.global_avg_pool(x)  # (batch, 512, 1)
        x = torch.flatten(x, start_dim=1)  # (batch, 512)

        # Capas fully connected
        x = self.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)

        return x
```

code 11 - RF matriz de confusión

```
cm = confusion_matrix(y_test, y_test_pred)
labels = y_test.unique()

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=labels,
yticklabels=labels)
```

```
plt.title('Matriz de Confusión')
plt.xlabel('Predicciones')
plt.ylabel('Valores Reales')
plt.show()
```

code 12 - RF piechart auxiliar de matriz de confusión

```
data = {
    "Tanker": {"Correcto": (217, 81), "Tug": (45, 17)},
    "Tug": {"Correcto": (263, 80), "Tanker": (60, 18)},
    "Cargo": {"Correcto": (39, 40), "Tanker": (44, 46)},
    "Passengership": {"Correcto": (29, 17), "Tug": (101, 66)}
}

fig, axes = plt.subplots(1, 4, figsize=(16, 4))

for ax, (category, values) in zip(axes, data.items()):
    total = 100
    used_percentage = sum(val[1] for val in values.values())
    other_percentage = total - used_percentage

    labels = [f"{key}\n{val[0]} casos ({val[1]}%)" for key, val in values.items()]
    labels.append("")  # Empty label for the black segment

    sizes = [val[1] for val in values.values()]
    sizes.append(other_percentage)

    colors = ["#4CAF50", "#FF5722", "black"]

    ax.pie(sizes, labels=labels, autopct="", startangle=90, colors=colors)
    ax.set_title(f"Predicción: {category}")

plt.tight_layout()
```

```
plt.show()
```

code 13 - RF classification report

```
from sklearn.metrics import classification_report
def visual_classification_report(y_test, y_test_pred):
  report = classification_report(y_test, y_test_pred, output_dict=True)
  report_df = pd.DataFrame(report).transpose()
  plt.figure(figsize=(10, 5))
  sns.heatmap(report_df, annot=True, cmap="RdBu", fmt=".2f", linewidths=.5,
  vmin=0, vmax=1)
  plt.title('Classification Report')
  plt.show()
```

code 14 - RF tree diagram

```
# Gráfica de los árboles
from sklearn import tree
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (20,4))
for index in range(0, 5):
    tree.plot_tree(RF.estimators_[index],
                   feature_names = X.columns,
                   filled = True,
                   impurity=False,
                   ax = axes[index]);


    axes[index].set_title('Estimator: ' + str(index), fontsize = 11)

# Seleccionar un solo árbol para mostrar
fig, ax = plt.subplots(figsize=(20, 12))  # Aumentar el tamaño de la figura
tree.plot_tree(RF.estimators_[0],  # Solo mostrar el primer estimador
               feature_names=X.columns,
               filled=True,
```

```
                impurity=False,
                ax=ax)

ax.set_title('Estimator: 0', fontsize=16)
plt.show()
```

code 15 - RF feature importances

```
# Obtener las importancias de las características
importances = RF.feature_importances_

# Filtrar las características con una importancia mayor a 0.0008
filtered_indices = [i for i, imp in enumerate(importances) if imp > 0.0008]

# Crear las listas de características y sus importancias filtradas
filtered_features = [X.columns[i] for i in filtered_indices]
filtered_importances = [importances[i] for i in filtered_indices]

# Graficar solo las características filtradas
plt.figure(figsize=(10, 6))
plt.barh(filtered_features, filtered_importances)
plt.xlabel("Feature Importances")
plt.ylabel("Features")
plt.title("Feature Importances Greater Than 0.0008")
plt.show()
```

code 16 - KNN normalización StandardScaler

```
## Normalizar los datos con StandardScaler
# Seleccionar solo las columnas numéricas
numerical_cols = df.select_dtypes(include=['number']).columns
# aplicar RobustScaler
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

code 17 - Extracción de características

```
def extract_audio_features(df_1):
  # Declarar listas vacias para generar df nuevas
  zcr_list = []
  rms_list = []
  spectral_centroid_list = []
  spectral_flux_list = []
  spectral_bandwidth_list = []
  mfcc_list = []
  delta1_mfcc_list = []
  delta2_mfcc_list = []

  # Sacar features de cada fila.
  for index, row in df_1.iterrows():
    audio_path = row['audio_path']
    try:
      y, sr = lb.load(audio_path)
      y = y - (min(y) + (max(y) - min(y)) / 2) # Para situar el 0 en el medio del
minimo y el maximo
      zcr, rms, spectral_centroid, spectral_flux, spectral_bandwidth, mfcc,
delta1_mfcc,delta2_mfcc = calcular_caracteristicas(y, sr, FRAME_SIZE,HOP_LENGTH)
      zcr_list.append(zcr)
      rms_list.append(rms)
      spectral_centroid_list.append(spectral_centroid)
      spectral_flux_list.append(spectral_flux)
      spectral_bandwidth_list.append(spectral_bandwidth)
      mfcc_list.append(mfcc)
      delta1_mfcc_list.append(delta1_mfcc)
      delta2_mfcc_list.append(delta2_mfcc)
    except Exception as e:
      print(f"Error al cargar el archivo {audio_path}: {e}")
      continue
```

```
  df_new = pd.DataFrame({
      'ship': df_1['ship'].to_list(),
      'date': df_1['date'].to_list(),
      'audio_path': df_1['audio_path'].to_list(),
      'zcr': zcr_list,
      'rms': rms_list,
      'spectral_centroid': spectral_centroid_list,
      'spectral_flux': spectral_flux_list,
      'spectral_bandwidth': spectral_bandwidth_list,
      'mfcc': mfcc_list,
      'delta1_mfcc': delta1_mfcc_list,
      'delta2_mfcc': delta2_mfcc_list
      })

  return df_new

df_features = extract_audio_features(df)
df_features
```

code 18 - Estandarización de dimensiones

```
df_features.applymap(lambda x: np.array(x).shape if isinstance(x, (list,
np.ndarray)) else type(x))

def split_column(df_, column_name):
  list_ = []

  for index,feature in df_.iterrows():
    feature_t = feature[column_name]
    row_ = {}
    for i in range(len(feature_t)):
      feature_t_row = feature_t[i]
```

```
      row_[f'{column_name}_{i+1}'] = [feature_t_row]
    list_.append(row_)

  df_temp = pd.DataFrame(list_)
  df_ = pd.concat([df_, df_temp], axis=1)

  df_ = df_.drop(column_name, axis=1)
  return df_

# Alineación de dimensiones de características
for col in ["zcr", "spectral_centroid", "spectral_bandwidth"]:
  df_features[col] = df_features[col].apply(lambda x: x.flatten() if isinstance(x,
np.ndarray) else x)
df_features.applymap(lambda x: np.array(x).shape if isinstance(x, (list,
np.ndarray)) else type(x))

df_features = split_column(df_features,'mfcc')
df_features = split_column(df_features,'delta1_mfcc')
df_features = split_column(df_features,'delta2_mfcc')
df_features

df_features_valores_solos = df_features
for col in df_features.columns:
  if col not in ["ship", "date", "audio_path"]:
    df_features_valores_solos = split_column(df_features_valores_solos,col)
    df_features_valores_solos[col] = df_features_valores_solos[col].apply(lambda x:
str(x).strip('[]'))
df_features_valores_solos.head()
```

code 19 - Carga de los datos

```
from google.colab import drive
drive.mount('/content/drive')
```

```
usr_folder = r'/content/drive/MyDrive/Big
Data/Datamining/Practica/2025_Datasets_Datamining' # Albert
usr_folder = r'/content/drive/MyDrive/Master/Data Mining/Pra´cticas Data
Mining/2025_Datasets_Datamining' # Wenjie

base_folder = usr_folder + '/dataset_lite/Train'
output_folder = usr_folder +  '/dataset_lite/Output'

def load_audio(input_folder, output_folder):
    data = []

    for ship_folder in os.listdir(input_folder): # Se accede a cada carpeta de los
4 barcos (Tug, Tanker, Passengership, Cargo)
        ship_folder_path = os.path.join(input_folder, ship_folder)
        if os.path.isdir(ship_folder_path): # Se comprueba que el path es correcto
            metadata_path = os.path.join(ship_folder_path, ship_folder.lower() +
"-metafile.txt")
            if os.path.exists(metadata_path):
                with open(metadata_path, "r", encoding="utf-8") as archivo: # Se
saca el archivo de metadatos
                    metadata_file = archivo.read()

            for date_folder in os.listdir(ship_folder_path): # Se accede a cada
dato registrado con fechas dentro de cada barco
                date = date_folder.split("-")[0] # De 20171106a-5 → 2017106a
                date = re.sub(r'[^0-9]', '', date)  # Elimina cualquier letra
                date_folder_path = os.path.join(ship_folder_path, date_folder)
                if os.path.isdir(date_folder_path):

                    for audio_file in os.listdir(date_folder_path): # Se accede a
los 5 audios de 4 segundos que hay en las carpetas de datos de las fechas
                        #print(audio_file)
```

```
                        if audio_file.endswith(".wav"):
                            audio_path = os.path.join(date_folder_path, audio_file)

                            data.append([ship_folder, audio_path, date,
metadata_path]) # Agregar datos al DataFrame
    return pd.DataFrame(data, columns=["ship", "audio_path", "date",
"metadata_path"]) # Crear el DataFrame

df = load_audio(base_folder, output_folder)
```

code 20 - Upsampling con SMOTE

```
from imblearn.over_sampling import SMOTE
from collections import Counter
smote = SMOTE()
X = df.drop(["ship", "date", "audio_path"], axis = 1)
y = df["ship"]
X_res, y_res = smote.fit_resample(X, y)
print(y_res)

sns.countplot(x=y_res)

y_res.value_counts()

df_to_save = pd.concat([X_res, y_res], axis = 1)
df_to_save.to_csv(output_folder + '/features_upsampling.csv', index=False)
```

code 21

```
def amplitude_mean_max_values(signal, frame_size, hop_length):
    amplitude_values = []
    # Iteramos sobre la señal de audio. Desde 0 hasta su longitud con el paso
definido por hop_length.
    for i in range(0, len(signal), hop_length):
```

```
        # De la señal tomamos una ventana de valores.
        # Empezando en el momento de tiempo "i" hasta "i" + la longitud de la
ventana (frame_size)
        window = signal[i:i+frame_size]
        # Agregamos el valor máximo de esta ventana a la lista de valores de la
amplitud
        amplitude_values.append(max(window))

    # Devolvemos la media de los valores de la envolvente de amplitud como
indicativo de intensidad del canto.
    return np.mean(amplitude_values)

def root_mean_squared(signal, frame_size, hop_length):
    rms = []
    # Iteramos sobre la señal de audio. Desde 0 hasta su longitud con el paso
definido por hop_length.
    for i in range(0, len(signal), hop_length):
        # Raíz de sumar los valores de la señal en esa ventana al cuadrado y
dividirlo por el tamaño de la ventana.
        rms_current_frame = np.sqrt(sum(signal[i:i+frame_size]**2) / frame_size)
        rms.append(rms_current_frame)
    return rms

def calcular_caracteristicas (y,sr,FRAME_SIZE,HOP_LENGTH):
  mfcc = lb.feature.mfcc(y=y, n_mfcc=13, sr=sr)
  #mfcc_scaled = np.mean(mfcc, axis=1)
  delta1_mfcc = lb.feature.delta(mfcc, mode='nearest')
  delta2_mfcc = lb.feature.delta(mfcc, mode='nearest', order=2)
  zcr = lb.feature.zero_crossing_rate(y=y)
  rms = root_mean_squared(y, frame_size=FRAME_SIZE, hop_length=HOP_LENGTH)
  amplitude = amplitude_mean_max_values(y, frame_size=FRAME_SIZE,
hop_length=HOP_LENGTH)
  spectral_centroid = lb.feature.spectral_centroid(y=y, sr=sr, n_fft=FRAME_SIZE,
```

```
hop_length=HOP_LENGTH)
  spectral_flux = lb.onset.onset_strength(S=lb.feature.melspectrogram(y=y, sr=sr,
n_fft=FRAME_SIZE, hop_length=HOP_LENGTH))
  spectral_bandwidth = lb.feature.spectral_bandwidth(y=y, sr=sr, n_fft=FRAME_SIZE,
hop_length=HOP_LENGTH)

  return (zcr, rms, spectral_centroid, spectral_flux, spectral_bandwidth, mfcc,
delta1_mfcc,delta2_mfcc)
```