Albert Ripoll Batlló

Máster Big Data La Salle, Barcelona

**Práctica Data Mining**

**Sep 12, 2024**

# **Índice**

* Introducción al problema. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **pág. 3**
* Explicación del pipeline. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **pág. 4**
  + 0- Esquema pipeline. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **pág. 4**
  + 1- Observar los audios y sus características. . . . . . . . . . . . . . . . . .  **pág. 4**
  + 2- Leer y cortar los audios. . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 6**
  + 3- Obtener las características por ventanas . . . . . . . . . . . . . . . . . .  **pág. 6**

### Zero Crossing Rate (ZCR). . . . . . . . . . . . . . . . . . . . . . . . pág. 7

### Root Mean Square (RMS). . . . . . . . . . . . . . . . . . . . . . . . pág. 7

### Amplitude\_mean\_values. . . . . . . . . . . . . . . . . . . . . . . . . pág. 7

### Mel-Frequency Cepstral Coefficients (MFCC). . . . . . . . . . . . pág. 8

### Delta 1 y Delta 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . pág. 9

### Spectral Centroid. . . . . . . . . . . . . . . . . . . . . . . . . . . . pág. 9

### Spectral Flux. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pág. 9

### Spectral Bandwidth. . . . . . . . . . . . . . . . . . . . . . . . . . . pág. 9

* + 4- Considerar audios de distintas longitudes. . . . . . . . . . . . . . . . .  **pág. 9**

### Audios demasiado cortos. . . . . . . . . . . . . . . . . . . . . . . pág. 9

### Características en forma de lista. . . . . . . . . . . . . . . . . . . pág. 10

* + 5- Modelado machine learning. . . . . . . . . . . . . . . . . . . . . . . .  **pág. 11**
    - **Random Forest (RF)**
    - **k-Nearest Neighbors (KNN)**
    - **Convolutional Neural Networks (CNN)**
* Análisis exploratorio y features escogidas. . . . . . . . . . . . . . . . **pág. 11**
  + Frecuencia de datos en cada especie. . . . . . . . . . . . . . . . . . .  **pág. 11**
  + Importancia de los atributos (Feature importances). . . . . . . . . . .  **pág. 13**
  + Correlación entre los atributos. . . . . . . . . . . . . . . . . . . . . . .  **pág. 14**
* Modelos de machine learning. . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 16**
  + Random Forest (RF) - Modelo 1. . . . . . . . . . . . . . . . . . . . . . .  **pág. 16**
    - **Lectura, separación y entrenamiento de los datos**. . . . . . . . . **pág. 16**
    - **Árboles de decisión**. . . . . . . . . . . . . . . . . . . . . . . . . . **pág. 17**
    - **Resultados: Precisión, Recall y F1-Score**. . . . . . . . . . . . . .  **pág. 18**
    - **Resultados: Matriz de confusión**. . . . . . . . . . . . . . . . . . .  **pág. 20**
    - **Resultados: Gráfica de árboles**. . . . . . . . . . . . . . . . . . . .  **pág. 21**
  + Random Forest (RF) - Modelo 2. . . . . . . . . . . . . . . . . . . . . . . .  **pág. 24**
  + k-Nearest Neighbors (KNN). . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 27**
    - **Lectura, separación y entrenamiento de los datos**. . . . . . . . .  **pág. 27**
    - **Resultados: Precisión, Recall y F1-Score**. . . . . . . . . . . . . .  **pág. 27**
    - **Resultados: Matriz de confusión**. . . . . . . . . . . . . . . . . . .  **pág. 28**
    - **Resultados: Precisión en función de la cantidad de vecinos**. . . .  **pág. 30**
  + Convolutional Neural Networks (CNN). . . . . . . . . . . . . . . . . . . . .  **pág. 30**
    - **Carga de espectrogramas e imports**. . . . . . . . . . . . . . . . . .  **pág. 30**
    - **Separación de los datos**. . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 31**
    - **Entrenamiento**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 33**
    - **Resultados**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 34**
* Resultados obtenidos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 36**
  + Random Forest (RF). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 36**
  + k-Nearest Neighbors (KNN). . . . . . . . . . . . . . . . . . . . . . . . . . . **pág. 39**
  + Convolutional Neural Networks (CNN). . . . . . . . . . . . . . . . . . . .  **pág. 41**
* Conclusiones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 42**
* Anexo código. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 45**
* Consulta y agradecimientos. . . . . . . . . . . . . . . . . . . . . . . . . . . .  **pág. 95**

Introducción al problema

El análisis acústico de las aves es una herramienta poderosa para comprender y estudiar la vida silvestre en entornos naturales. Permite obtener información valiosa sobre la presencia, distribución y comportamiento de las aves en su hábitat. Sin embargo, el análisis manual de los audios es una tarea que requiere la experiencia de expertos y se hace imposible en grandes volúmenes de datos. Por ese motivo surge la necesidad de desarrollar un sistema automatizado capaz de detectar y clasificar los cantos de las aves de forma precisa y eficiente. Este sistema tendría aplicaciones en la monitorización de la biodiversidad, la conservación de especies y la gestión de áreas protegidas.

El objetivo del presente trabajo es diseñar e implementar un sistema de aprendizaje automático (Machine Learning) que pueda clasificar automáticamente los sonidos de aves de diferentes especies. Para ello, se utilizará el conjunto de datos de Western Mediterranean Wetlands Bird Dataset. Este dataset está [disponible para su descarga](https://zenodo.org/records/7505820#.Y8U4f3bMKUk) y contiene grabaciones de audio que tienen etiquetados los segundos de canto y la especie de ave al que corresponde. El trabajo requiere: analizar los datos, usar un algoritmo de Machine Learning, usar un ensemble y usar un algoritmo de Deep Learning.

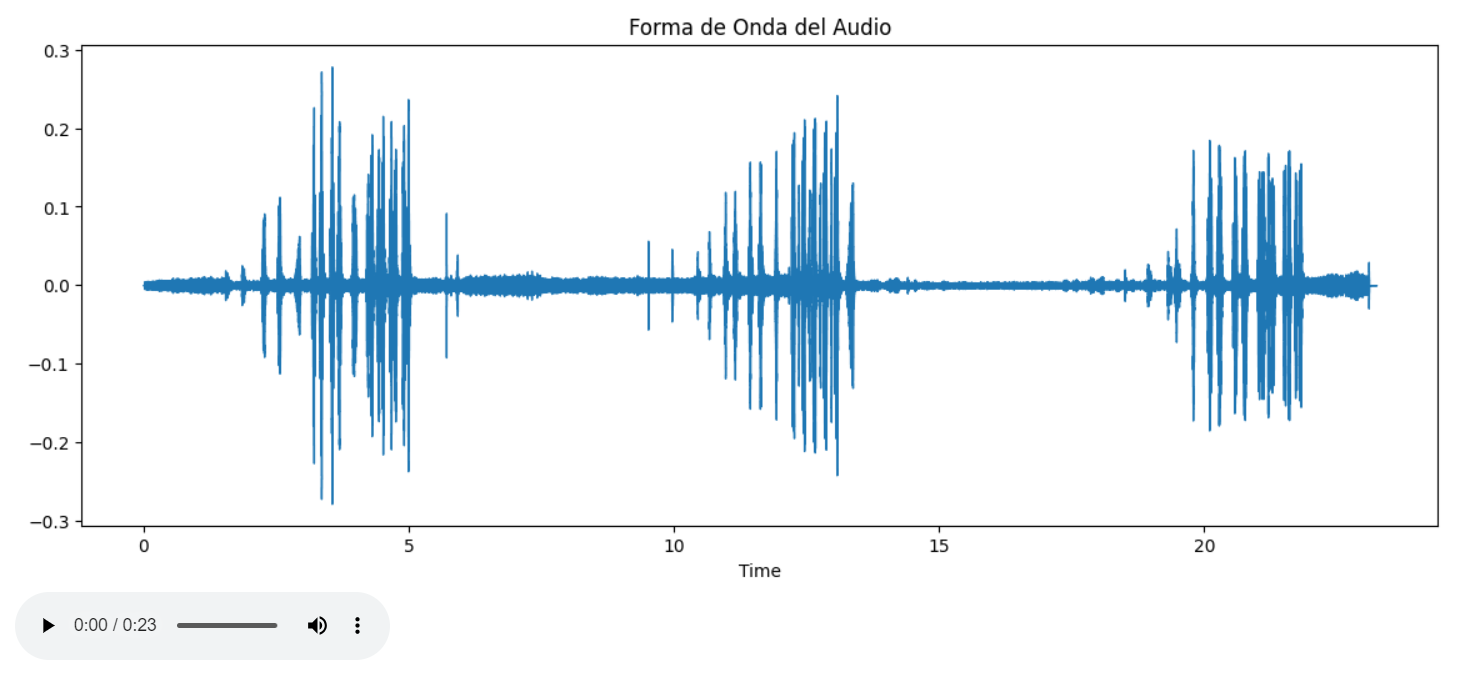
Explicación del pipeline

# 0- Esquema pipeline

# 

# 1- Observar los audios y sus características

Se observa como son los datos de audio y qué características tienen para ser analizados.

**code 1**

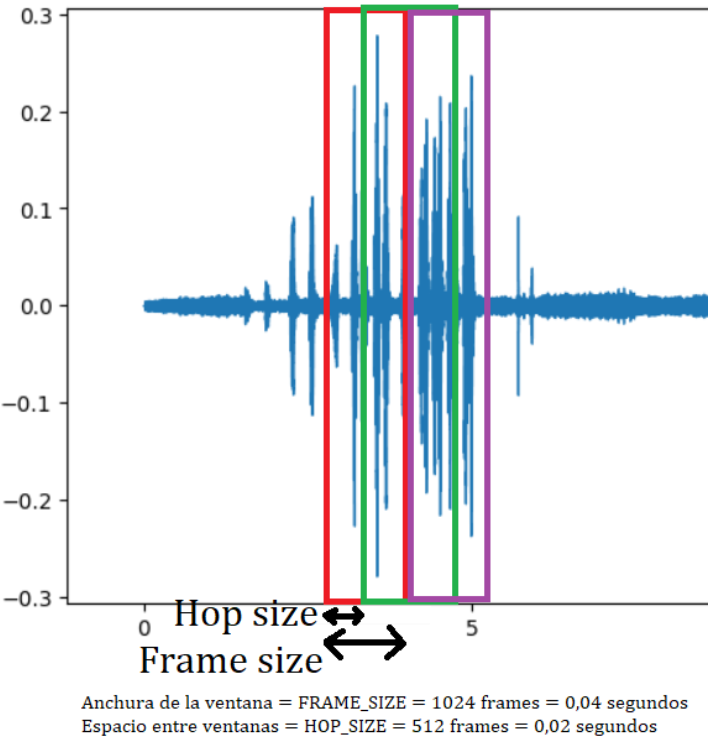
Se determina que el análisis de características del audio se hará con ventanas de muestreo del audio. Hay dos conceptos clave:

• HOP\_LENGTH o HOP\_SIZE: Espacio entre ventanas. Lo definiremos de 512 frames, o sea 0,02 segundos.

• FRAME\_SIZE: Anchura de la ventana. Lo definiremos de 1024 frames, o sea 0,04 segundos.

De esta forma entendemos que la precisión del análisis del audio es de cada 0,04 segundos y el muestreo es de tal forma que se asegura que cualquier valor es analizado como mínimo por 2 ventanas distintas.

De forma gráfica, pero no a escala, este concepto se puede representar así:



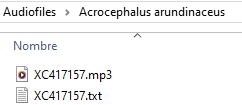
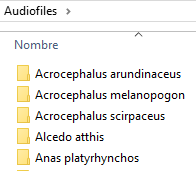
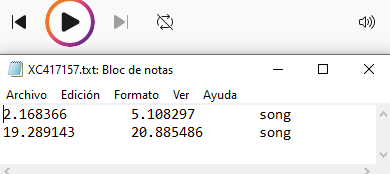
De este modo, el análisis se hará según las características dentro de cada “frame” y se irán muestreando según cada “hop”.

# 

# 2- Leer y cortar los audios

Los Audiofiles [vienen](https://zenodo.org/record/7505820#.Y8U4f3bMKUk) comprimidos en un ‘.zip’ donde hay una carpeta para cada ave. Dentro de esa carpeta hay dos tipos de ficheros. Uno con las vocalizaciones de las aves en formato ‘.mp3’ y otro un fichero ‘.txt’ que indica momentos donde se inician y finalizan los cantos que registra el audio.

Carpeta para cada ave Par de ficheros dentro Fichero .mp3 y .txt



→ →

Entonces, lo primero que se requiere hacer es cortar los trozos de los audios donde hay canto ya que es la parte de toda la grabación que interesa tratar. Se realizan los cortes de los “Audiofiles” y se guardan como “Audio\_clips” con la misma disposición y nombre a excepción que a los nombres de los archivos se les añade \_1, \_2, \_3 para indicar distintos clips el mismo audio.

Audiofiles → Audio\_clips **code 2**

# 3- Obtener las características por ventanas

Como se había comentado en el punto 1, las características de los audios se analizan por ventanas determinadas por:

• HOP\_LENGTH o HOP\_SIZE: Espacio entre ventanas. Lo definiremos de 512 frames, o sea 0,02 segundos.

• FRAME\_SIZE: Anchura de la ventana. Lo definiremos de 1024 frames, o sea 0,04 segundos. **code 3**

De esta forma entendemos que la precisión del análisis del audio es de cada 0,04 segundos y el muestreo es de tal forma que se asegura que cualquier valor es analizado como mínimo por 2 ventanas distintas.

Las características que son interesantes de analizar de los audios y que los hace distinguibles unos de los otros son las siguientes. En este paso del pipeline, se busca analizar y extraer estas características para los audios previamente cortados. Tras los análisis, se generan dataframes *df\_ml* y *df\_repetir\_ml* (en el caso de RF Modelo 2) que se guardan en Drive en formato .parquet preparados para ser usados posteriormente en el paso de *machine learning*.

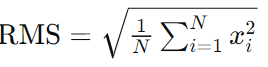
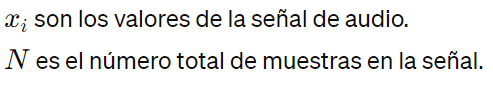
### Zero Crossing Rate (ZCR)

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). En este caso, el ZCR se calcula para el audio entero y no por ventanas. **code 3**

### 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 3**

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:

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. Entonces, hace una media para cada ventana. Después, al final, devuelve la media de todas las ventanas.

**code 3**

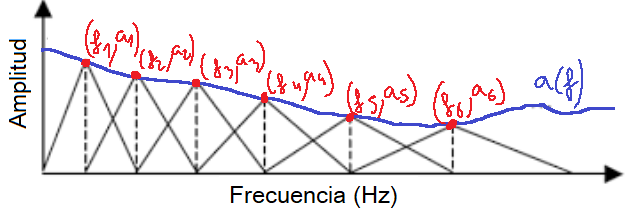
### Amplitude\_mean\_values

Otra forma de parametrizar la intensidad del canto 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. **code 3**

### 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](https://www.youtube.com/watch?v=4_SH2nfbQZ8&list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0&index=20) 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 transformada 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 (ak) de la frecuencia como características.

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

### 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.

### 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.

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

### 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.

### 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.

# 4- Considerar audios de distintas longitudes

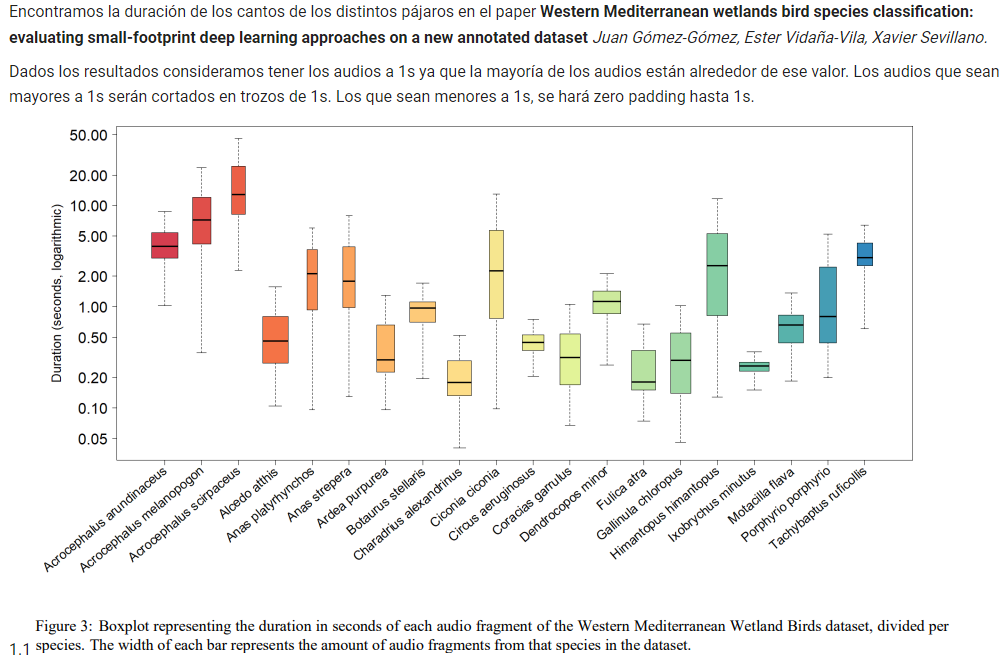
Llegados a este punto, nos damos cuenta que los audios analizados tienen distintas longitudes en cuanto a duración temporal se refiere. Se encuentran varios problemas:

### Audios demasiado cortos

Se encuentra que algunos de los audios analizados son más pequeños que las ventanas de análisis que se les aplica. Por eso es que descartamos estos audios. if len(y) < FRAME\_SIZE\*2: continue

### Características en forma de lista

Las características que se extraen mencionadas en el punto 3 del pipeline son valores únicos (por lo general medias) o son valores en lista o matriz (caso de mfcc, delta1 y delta 2). Esto dará muchos problemas posteriormente al aplicar modelos ya que necesitamos tener un dataframe con cada celda un valor y no listas. También el hecho de que los audios sean de distintas duraciones hace que unos tengan mayor número de ventanas analizadas y datos que otros y que los que no tienen datos tengan valores nulos, que es un problema añadido y difícil de lidiar de forma satisfactoria.



De esta forma todos los audios que se analizen tendrán la misma duración (1 segundo), con lo que este factor será constante entre todos los audios y no añadirá variaciones indeseadas. Tras este pipeline, obteniendo las características por ventanas del punto 3, se guardan los dataframes df\_mfcc y df\_delta. Para la mfcc **code 4, 5 y 6**. Para la delta **code 7, 8 y 9.**

# 5- Modelado de machine learning

Para el modelado, leemos el dataframe de las características que se han podido analizar y se añaden los nombres de las especies y el nombre de archivo. **Code 10, 11 y 12**

Entonces, se exploran los datos y se lleva a cabo el modelado de los siguientes modelos:

### Random Forest (RF)

### k-Nearest Neighbors (KNN)

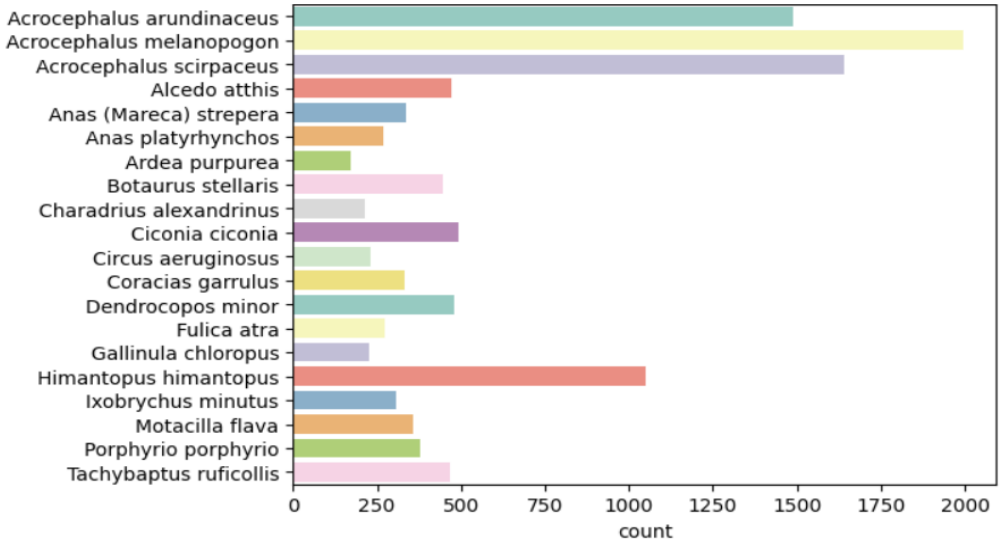
### Convolutional Neural Networks (CNN)

Análisis exploratorio y features escogidas

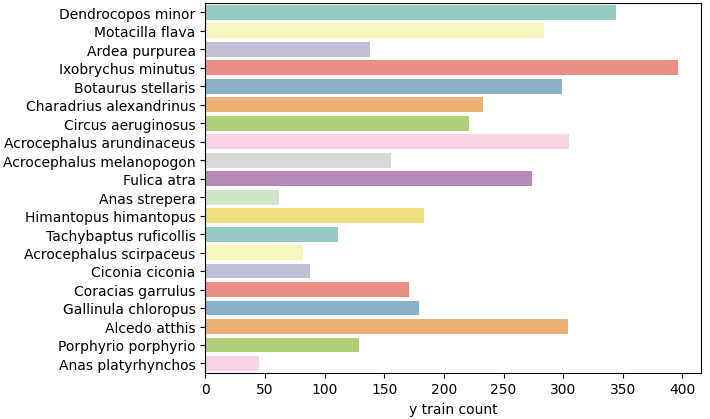
# Frecuencia de datos en cada especie

En un primer análisis exploratorio vemos que hay algunas clases (Asas strepera, Acrocephalus scirpaceus, Anas platyrhynchos, Ciconia ciconia) que son menos frecuentes que el resto. Esto puede generar que los modelos tiendan a entrenar peor dichas clases debido a tener menos casos y por lo tanto predecir erróneamente. Una posible solución al problema sería aplicar la técnica de Up Sampling. Esta técnica genera datos de las clases minoritarias copiando el dato y aplicando pequeñas modificaciones para que no sean exactamente iguales. La otra opción sería la técnica opuesta, Down Sampling, la cual reduce las clases mayoritarias hasta igualarse con el resto de las clases.

En nuestro caso vemos que la diferencia entre el número de casos entre una clase y otra no es tan grande con respecto a trabajos similares.

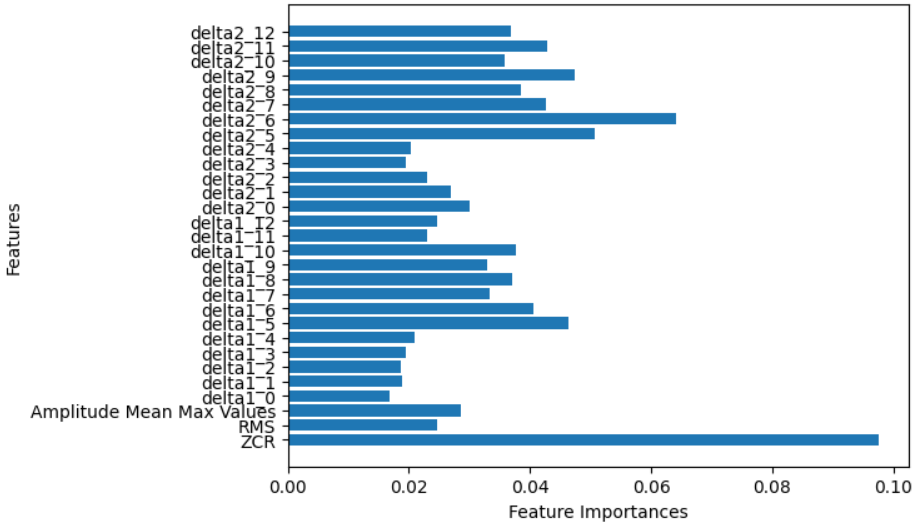


↑ Figura 4. Balance de clases sin normalizar la longitud del audio

↓ Figura. Balance de clases normalizando la longitud del audio a 1s

# Importancia de los atributos (Feature importances)

Al analizar la importancia de unas características u otras para predecir la correcta clasificación del audio a la especie, vemos que las características finalmente procesadas cuentan todas con bastante importancia. Con mayor importancia encontramos la característica de ZCR con diferencia, seguida delta2\_6, delta2\_5 y delta 2\_5. Se observa que las que menor importancia tienen son los coeficientes entre 0 y 4 de delta junto con delta 1 de 11 a 12.

**code 13**

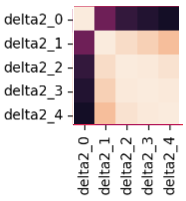
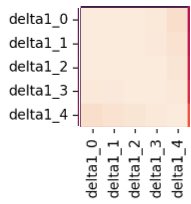
# 

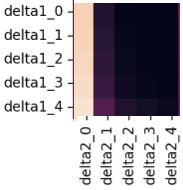
# Correlación entre los atributos **code 14**

# En cuanto a las dependencias entre las variables podemos comentar lo siguiente. Si dos variables están correlacionadas se puede entender que una complementa a la otra y no es necesario usar ambas ya que podríamos prescindir de una al tener la otra información similar.

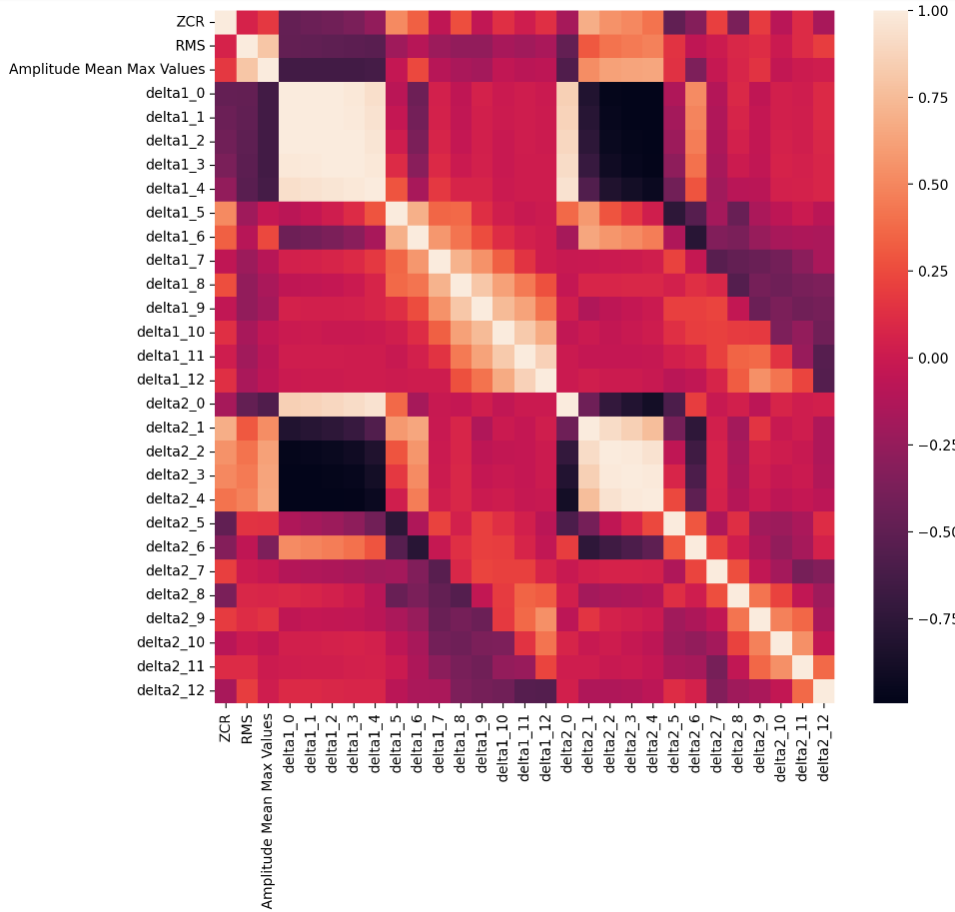
* Las variables que están **correlacionadas** (aportan poca información nueva una de la otra) son:

Los coeficientes entre 0 y 4 de Delta1 y Delta2. Concretamos sus dependencias:

* Delta1\_0, Delta1\_1, Delta1\_2, Delta1\_3 y Delta1\_4 que se correlacionan fuertemente (r~-0,75 o r~1 en el caso de delta2\_0) con Delta2\_0, Delta2\_1, Delta2\_2, Delta2\_3 y Delta2\_4.
* Delta2\_0, Delta2\_1, Delta2\_2, Delta2\_3 y Delta2\_4 que tienen correlaciones fuertes entre ellas.
* Delta1\_0, Delta1\_1, Delta1\_2, Delta1\_3 y Delta1\_4 que tienen correlaciones muy fuertes (r~1) entre ellas.



* Delta1\_0, Delta1\_1, Delta1\_2, Delta1\_3 y Delta1\_4 que tienen correlaciones muy fuertes (r~1) con Delta2\_0, Delta2\_1, Delta2\_2, Delta2\_3 y Delta2\_4.
* Las variables que **no** están **correlacionadas** (se complementan aportando información nueva) son las demás. Lo podemos observar en un mapa de calor global donde claramente se diferencian las cuatro zonas anteriormente mencionadas.





El hecho que las primeras deltas estén correlacionadas es plausible y lógico si pensamos de dónde provienen estos coeficientes. Estos representan las variaciones de 1r y 2n orden entre los coeficientes mfcc. Los coeficientes mfcc son las variaciones de las amplitudes de las frecuencias bajas y la selección de coeficientes mfcc se realiza por ventana de frecuencia logarítmica. Por lo que los 4 primeros valores son valores mfcc muy cercanos y similares, con poca variación (delta) entre ellos sea del audio que sea.

Modelos de machine learning

# RF (Random Forest) - Modelo 1

El Random Forest es un modelo de machine learning de tipo *Ensemble* de distintos árboles de decisión.

**Lectura, separación y entrenamiento de los datos**

* Lectura e imports: **code 15**
* Separación de los datos: Para realizar la separación de los datos se tiene en cuenta que en ambos conjuntos test y train no puede haber datos del mismo audio (mismo audio que ha sido cortado en pedazos de 1 segundo). Es decir se tiene que evitar que haya segmentos de 1 segundo del mismo audio en los subconjuntos test y train. Porque sino podríamos estar situando audios iguales en ambos conjuntos. Si no tuviéramos en cuenta este hecho, eso supondría viciar los datos ya que sería añadir “best cases”. Estaríamos clasificando el mismo pájaro todo el rato. Por eso es necesario e importante hacer la separación de esta forma. Una forma de evitarlo es establecer que todos los audios del mismo pájaro vayan o bien todos a test o bien todos a train. En primera instancia se ha procedido así en este modelo 1. Sin embargo, esta forma de proceder es incorrecta y sin sentido ya que si lo pensamos es como si estuviéramos entrenando los datos con audios de “ocas” y después estuviéramos intentando predecir los audios de “gallinas” a partir de ese modelo entrenado con “ocas”. Dado que el modelo estaría entrenado con unos tipos de especies, no sería bueno ni coherente que estuviera prediciendo otros tipos de especies. Sin embargo, se considera interesante incluir este modelo en el trabajo para ver sus resultados. En el primer párrafo del apartado de resultados se discute y se explica claramente y a nivel conceptual por qué los resultados del modelo no tienen sentido, hecho que es interesante entender y discutir.

Al programarlo se ha usado la estrategia de "GroupShuffleSplit" de sklearn.model\_selection. Ya que permite dividir los datos manteniendo los grupos juntos. En este caso, cada directorio será un grupo, y nos aseguraremos de que ningún directorio esté presente tanto en el conjunto de entrenamiento como en el de prueba. **code 16**

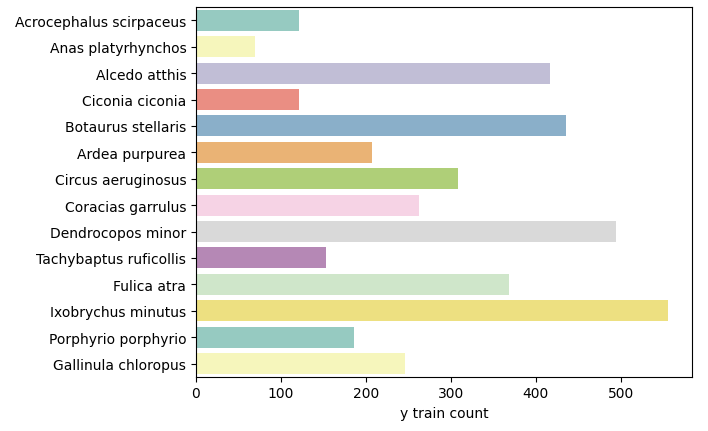
Tamaño del conjunto de entrenamiento: (3946, 29) (3946,)

Tamaño del conjunto de prueba: (1775, 29) (1775,)

Directorios en el conjunto de entrenamiento: ['Acrocephalus scirpaceus' 'Anas platyrhynchos' 'Alcedo atthis' 'Ciconia ciconia' 'Botaurus stellaris' 'Ardea purpurea' 'Circus aeruginosus' 'Coracias garrulus' 'Dendrocopos minor' 'Tachybaptus ruficollis' 'Fulica atra' 'Ixobrychus minutus' 'Porphyrio porphyrio' 'Gallinula chloropus']

Directorios en el conjunto de prueba: ['Acrocephalus arundinaceus' 'Anas strepera' 'Acrocephalus melanopogon' 'Charadrius alexandrinus' 'Himantopus himantopus' 'Motacilla flava']

Intersección entre listas = CONJUNTO VACÍO



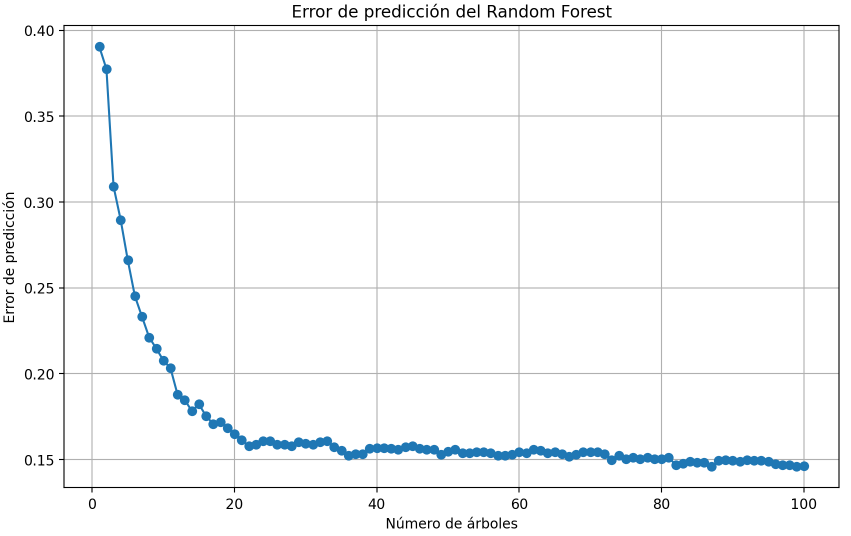
En una segunda instancia se ha considerado que no tiene sentido entrenar los datos con unas especies de pájaros en concreto para poner a prueba el modelo (testear) para clasificar otros pájaros distintos. Este caso se ha corregido en el Modelo 2.

* Entrenamiento: **code 17**

**Árboles de decisión code 17**

Se ha realizado un barrido de 1 a 100 números de ‘árboles de decisión’ usados para el RF evaluando el error de predicción para cada caso. A mayor número de clasificadores (árboles), el error de la predicción es menor. Pero a mayor número de árboles el coste computacional es mayor.

Como tenemos 20 especies, se fija max\_depth=20 y como tenemos capacidad computacional suficiente para el caso, seleccionamos 200 clasificadores aunque como hemos visto en el gráfico no harían falta tantos. n\_estimators = 200

****

**Resultados: Precisión, Recall y F1-Score code 18**

Es importante constatar que en el caso del modelo 1 no es posible calcular la precisión ya que seguro que es imposible sacar un verdadero positivo ya que el modelo no ha estado entrenado para decir positivo en los modelos de test. Solo ha estado entrenado para decir positivo en los modelos entrenados de train. El modelo que ha estado entrenado con un tipo de especie nunca dirá o predecirá datos con otras especies ya que el modelo ni siquiera sabe que existen otras especies posibles. Es decir, un dato testeado nunca acabará siendo constatado como la especie que es. Sería interesante como ejercicio observar qué errores hay en en el código 16 en donde la separación de los datos alguna especie que se pretendía que fuera solo de test se ha colado en el conjunto de entrenamiento o bien qué errores hay en el código 18 que genera los resultados que haya hecho posible sacar algún resultado de precisión.

Se verifica el modelo RF con las métricas de Precisión, Recall y F1-Score. Seguidamente se describen las métricas del modelo y se ponen unos ejemplos de resultados de este modelo.

Una precisión alta significa que hay pocos falsos positivos, es decir, indica que la probabilidad de que la predicción sea correcta es alta. Una precisión baja implica que hay bastantes falsos positivos, es decir, muchas predicciones positivas son incorrectas.

Se obtiene una precisión en los datos train de 0,81 y en los datos test de 0,66.

****

* . El "recall" (también sensibilidad o tasa de verdaderos positivos) es la proporción de ejemplos positivos que fueron correctamente identificados por el modelo respecto al total de ejemplos positivos en los datos de prueba. El “recall” es útil cuando el objetivo principal es identificar todos los casos positivos, incluso a costa de clasificar incorrectamente algunos ejemplos negativos como positivos.

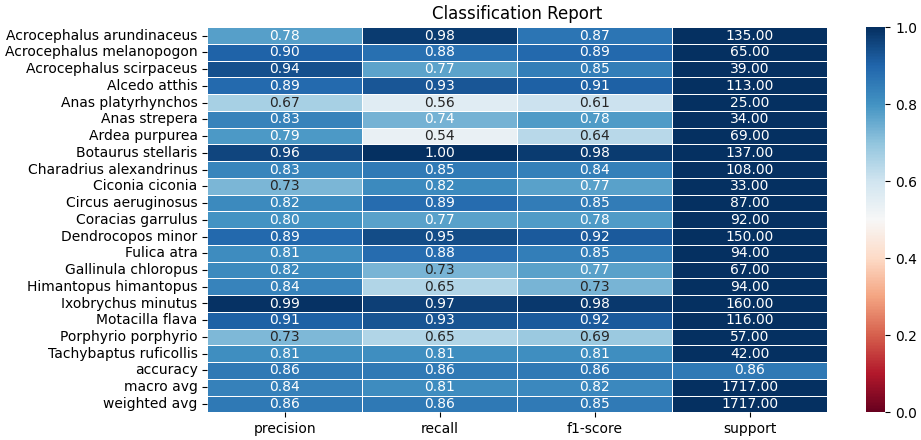
Una recuperación alta indica que el modelo “recupera” correctamente todos los casos positivos. Es decir, indica que el modelo es capaz de detectar los casos que son positivos, se pierde pocas instancias positivas. Una recuperación baja indica que pocos casos que eran positivos fueron detectados correctamente por el modelo. Es decir, el modelo no es capaz de detectar los casos positivos, se pierde bastantes instancias positivas.

* . El "F1 score" es una medida que combina precisión y "recall" en un solo valor. Es especialmente útil cuando hay un desequilibrio entre las clases en los datos como es un poco el caso.

Un f1-score alto indica que tanto la precisión como la recuperación son altas y estan bien equilibradas. Un f1-score bajo indica que la precisión como la recuperación son bajos o bien que están desequilibradas. O sea, el modelo tiene muchos falsos positivos y/o muchos falsos negativos.

* "Support" es el número de muestras verdaderas que pertenecen a cada clase.

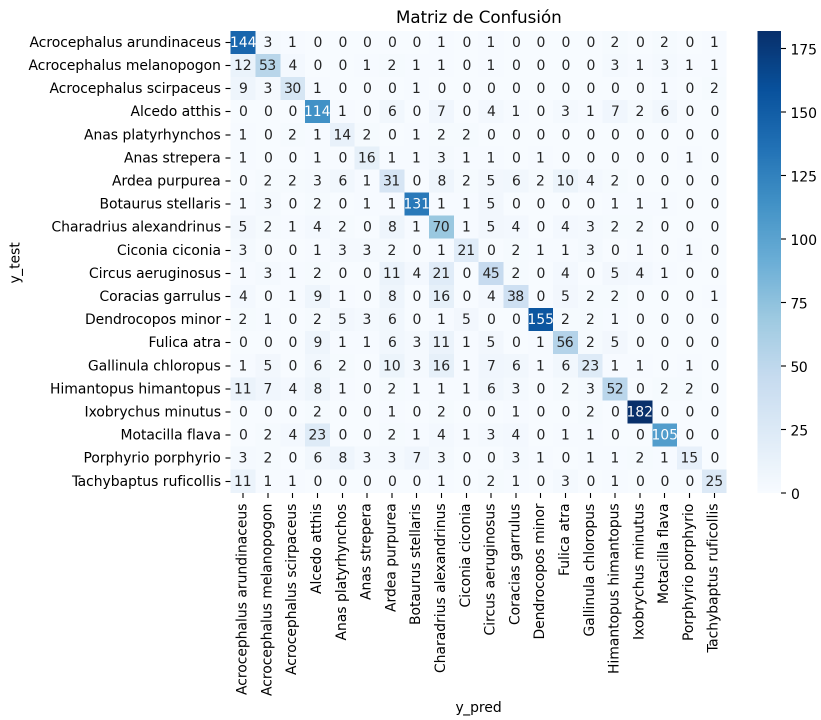
Se obtienen valores muy satisfactorios de Precision, Recall y F1 score. Todos estando por encima de los 0,5 y solo 4 valores entre todas las especies y métricas estando por debajo de 0,65 en Anas playtrhynchos y Ardea purpurea, que son las especies con menos datos.



**Resultados: Matriz de confusión code 19**

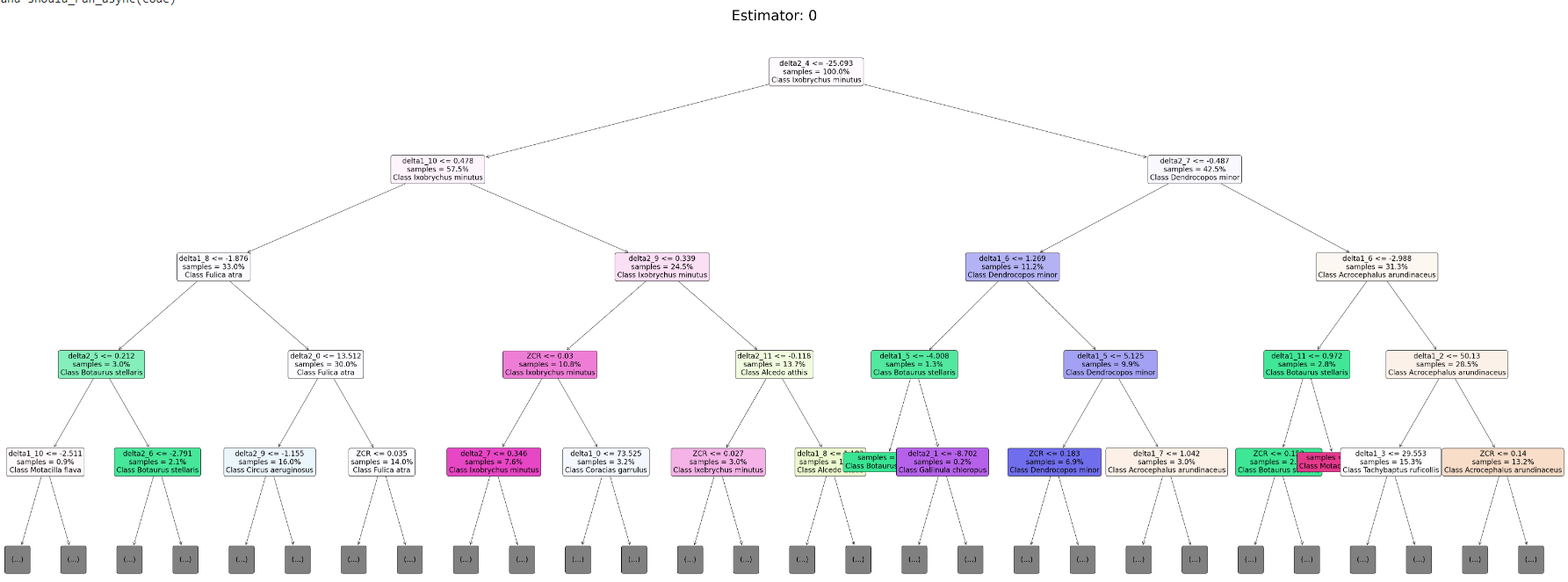
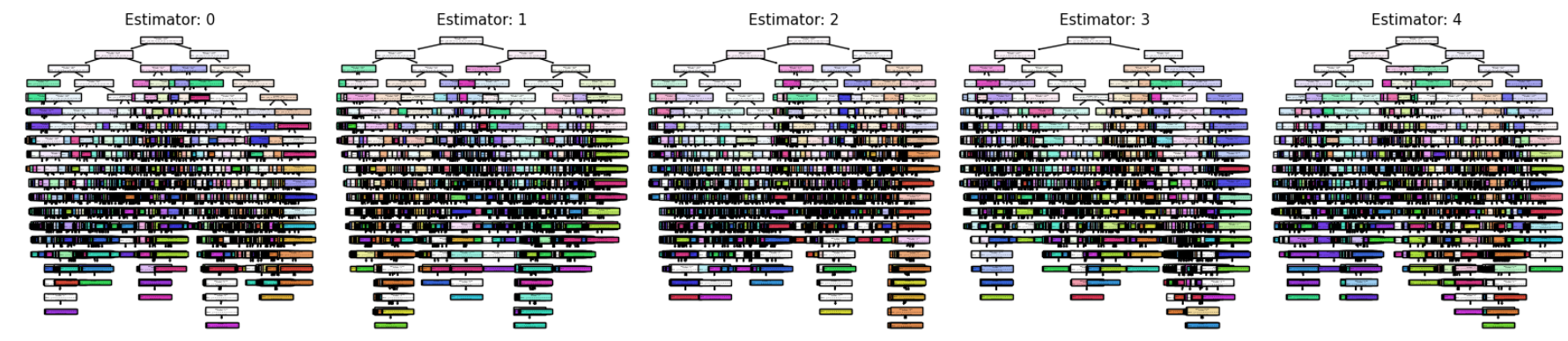
La matriz de confusión compara las predicciones del modelo (y\_pred) con las especies reales de los datos de prueba (y\_test). Nos muestra la cantidad de datos que fueron precedidos en cada modo. Observando nuestra matriz observamos mayor cantidad de valores en la diagonal definida, hecho que corresponde a predicciones verdaderas. También se observan las cantidades que fueron erróneamente predichas y donde fueron mal predichas.

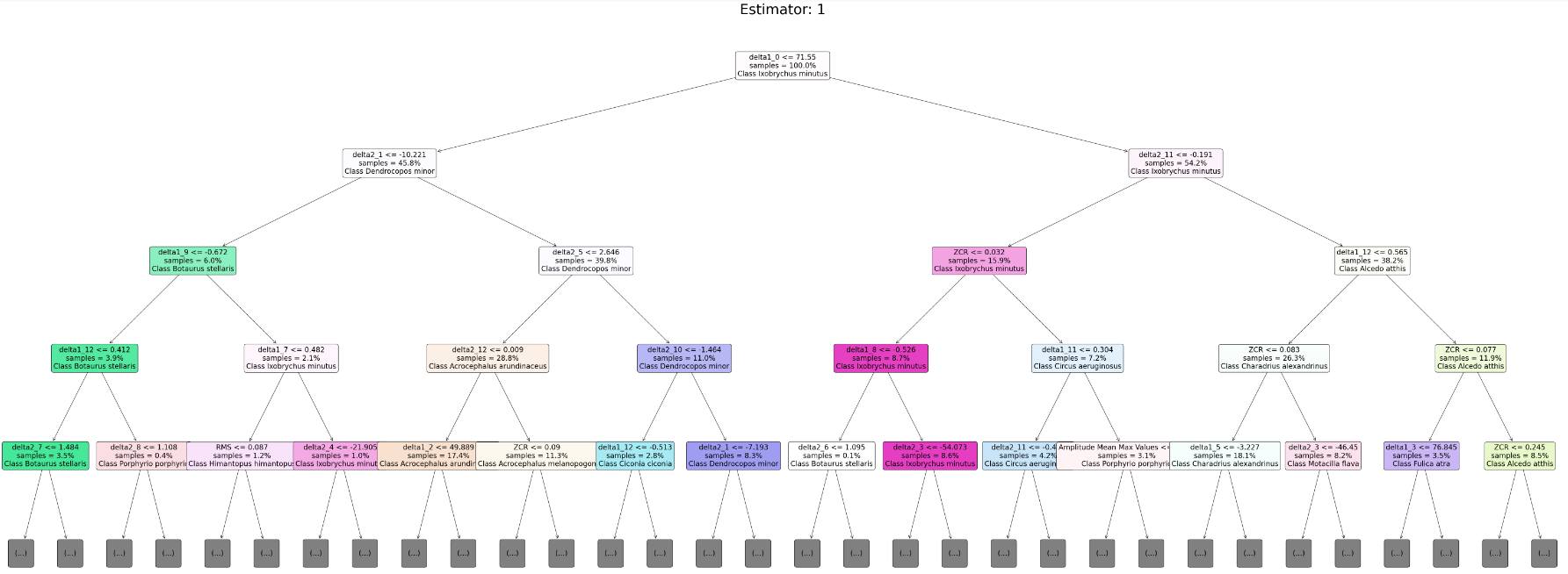
En concreto las especies como *Circus aeruginosus*, *Dendrocopos minor* y *Fulica* atra tienen una alta cantidad de predicciones correctas. Sin embargo, hay ciertas confusiones. Por ejemplo, *Gallinula chloropus* es confundida varias veces con *Fulica atra*, y *Moticilla flava* con *Acrocephalus melanopogon*.

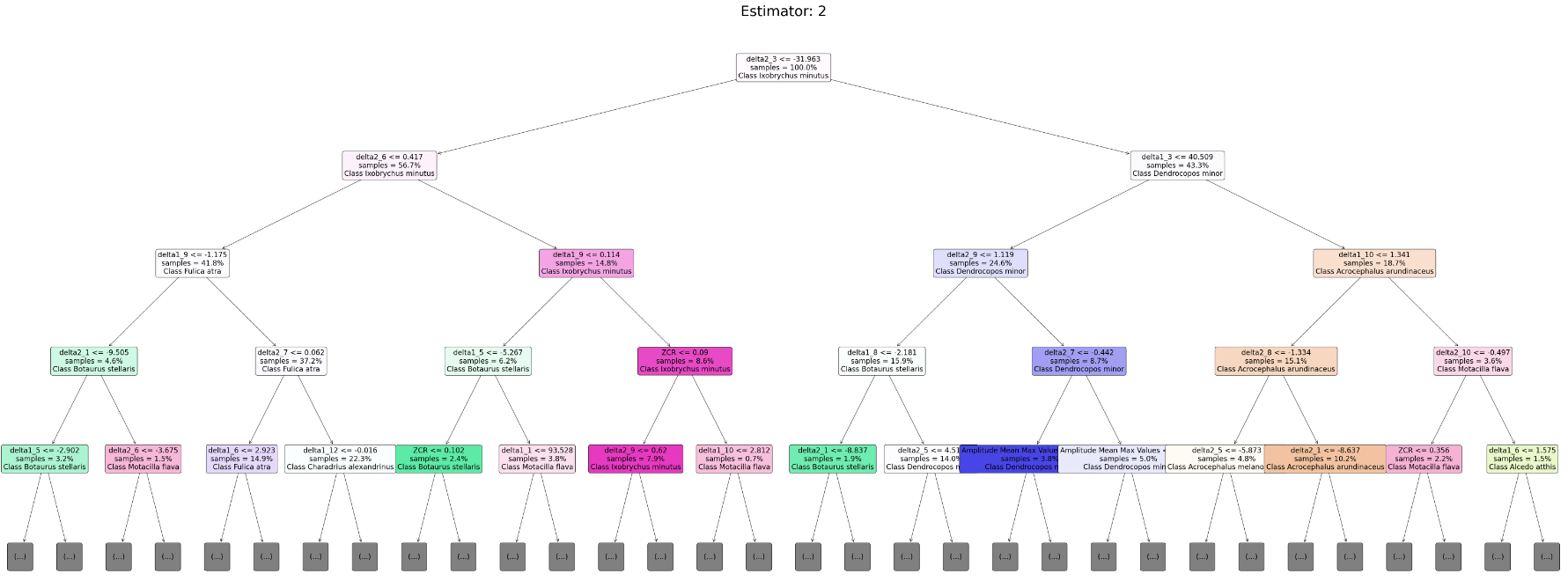
****

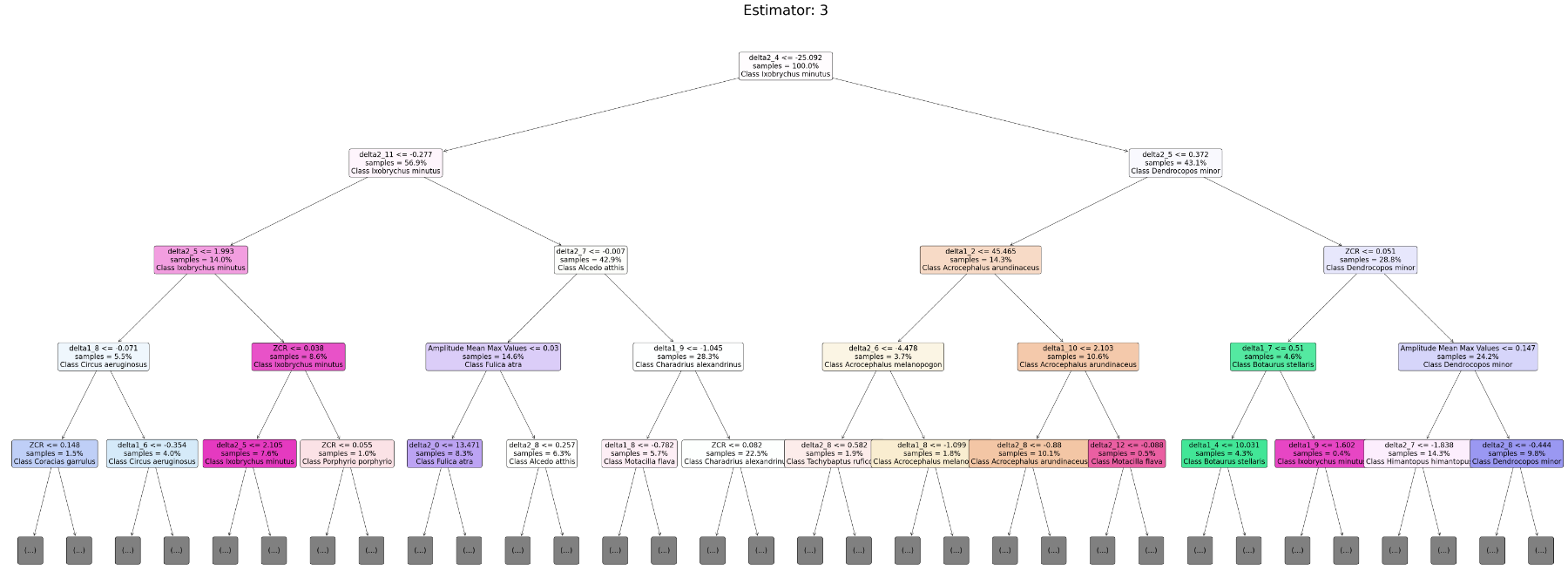
**Resultados: Gráfica de árboles code 20**

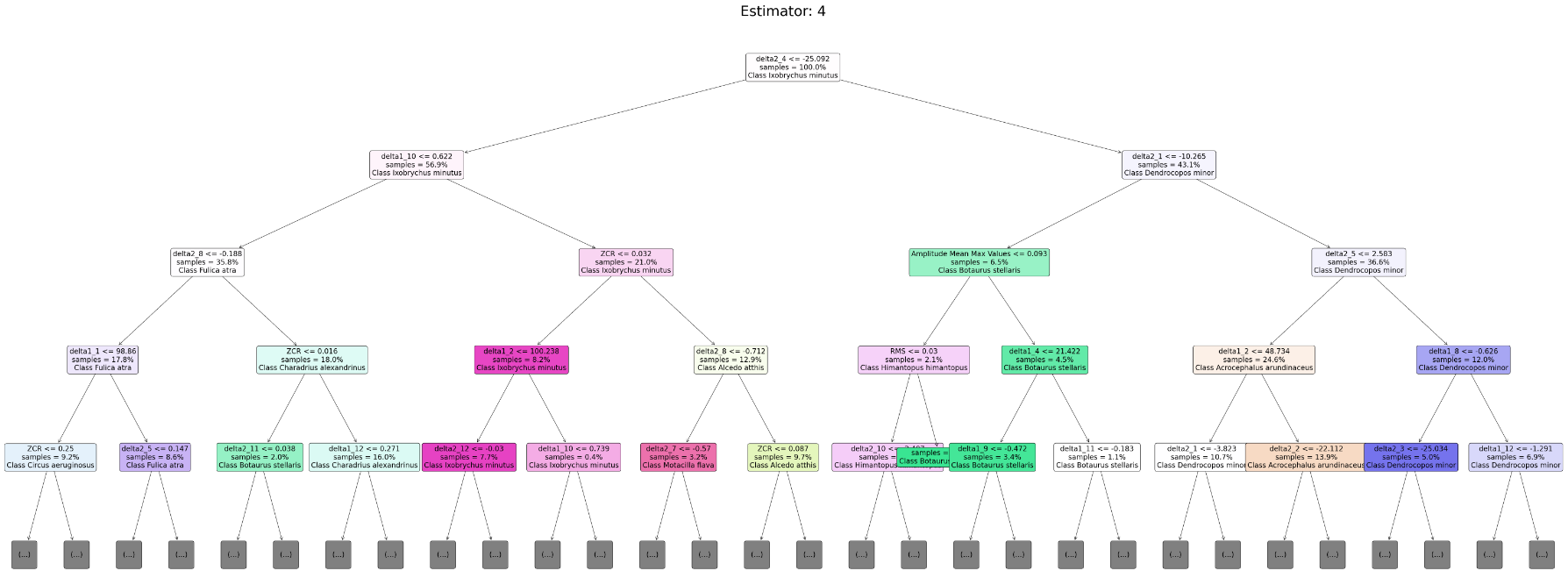
Finalmente la gráfica de los árboles nos ilustra la estructura de un árbol de decisión. Cada nodo del árbol representa un atributo y cada borde conecta nodos representando las decisiones tomadas basadas en esa característica (eg: si la característica es mayor qué tal valor, entonces es esta especie). Las hojas del árbol representan las clases a las que pertenecen los datos. Los diagramas árbol se realizan para distintos estimadores que, en el contexto de los modelos de ensamble como Random Forest, los estimadores son un modelo de aprendizaje individual dentro del ensamble. 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.













# RF (Random Forest) - Modelo 2

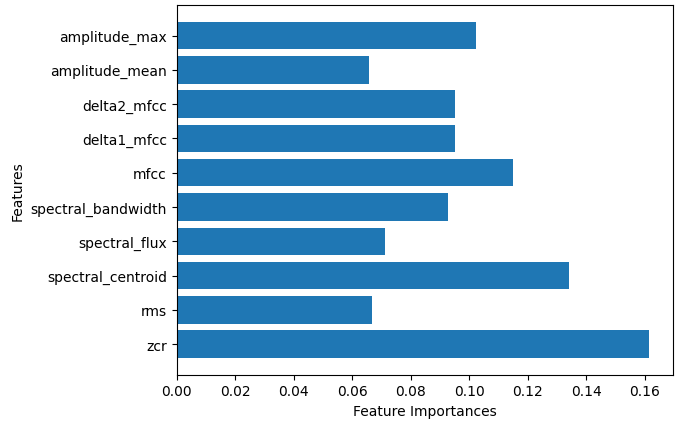
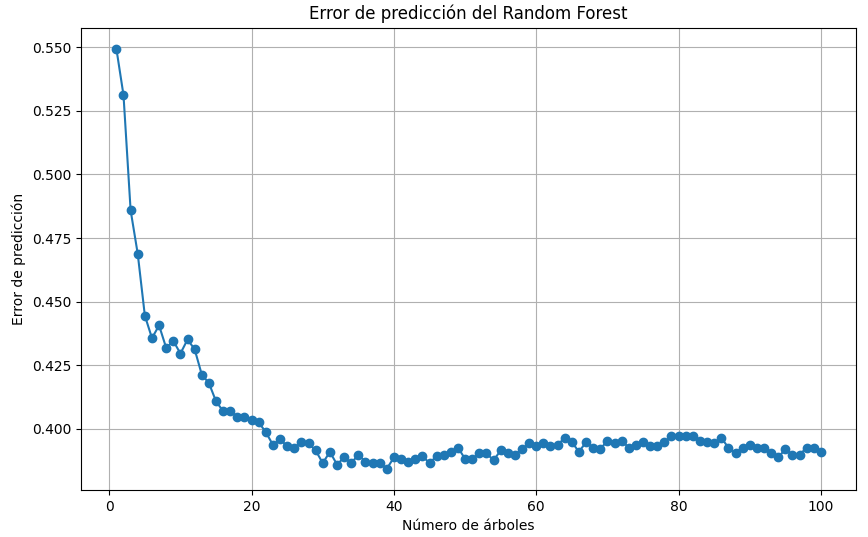
Siguiendo ya las explicaciones y pasos hechos en el modelo 1, en este modelo se modifican algunos pasos.

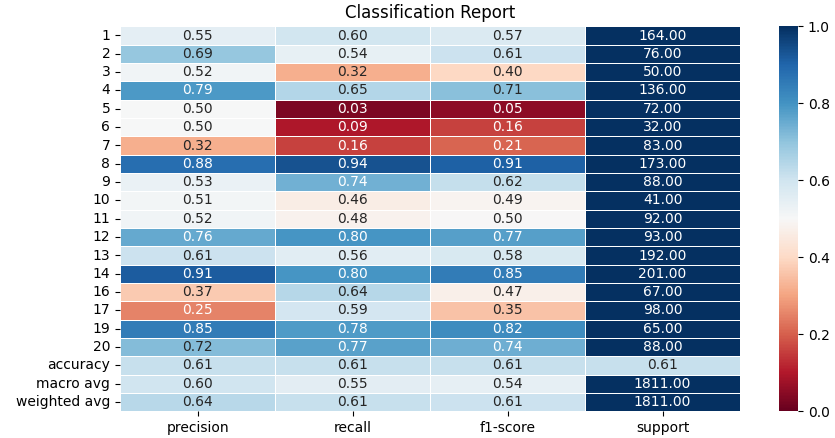
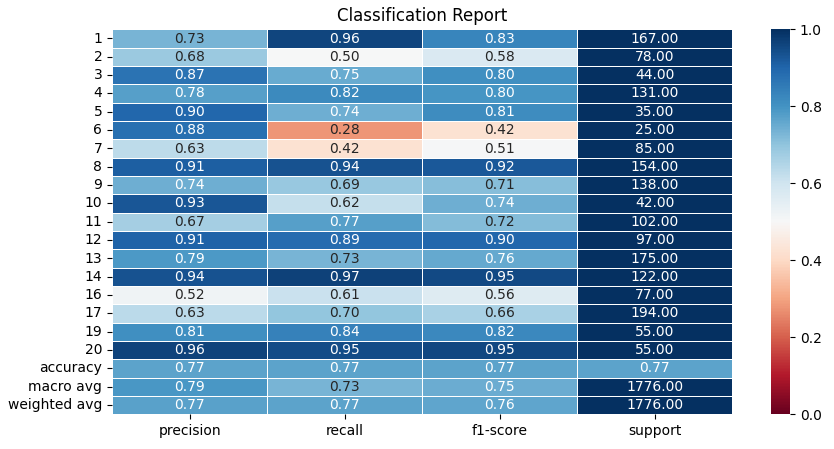
Tal como sugirió Ester Vidaña Vila (autora del paper de la figura 3), al comentar sobre el presente trabajo, se realiza un segundo modelado de Random Forest que no realiza zero padding para los audios menores a 1 segundo. En cambio, a diferencia del “Modelo 1”, este “Modelo 2” lo que hace es replicar y acumular los audios que son menores a 1 segundo hasta que lleguen a tener esta longitud. El código usado es similar al del modelo 1 a diferencia de la parte donde se define hacer repetición del audio en vez de zero padding (**code 37**):

if len(y) < max\_length \* sr: # Si el audio es más pequeño que 1 segundo, repetir repeats = (max\_length \* sr) // len(y) + 1 # Número de repeticiones necesarias

y = np.tile(y, repeats)[:max\_length \* sr] # Repetir y recortar

Se cambia el nombre de archivo en vez de df.parquet a df\_repetir.parquet y de df\_ml.parquet a df\_repetir\_ml.parquet en el código 10, 11 y 15 (**code 38**). También el modelo 2 difiere del modelo 1 porque no se separa train y test por tipo de pájaro sino que se hace por tipo de audio (**code 39**). Finalmente, se aprovechó para añadir nuevas features que no fueron añadidas anteriormente como los spectral bandwidth, flux y centroids, la mfcc mean y el amplitude max value. A continuación se muestran los resultados (**code 40**).



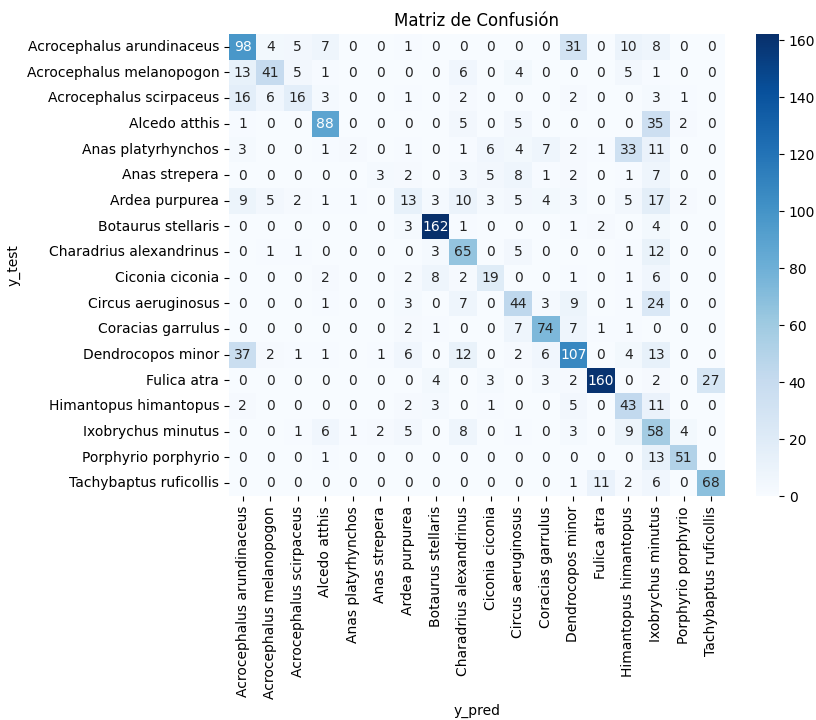
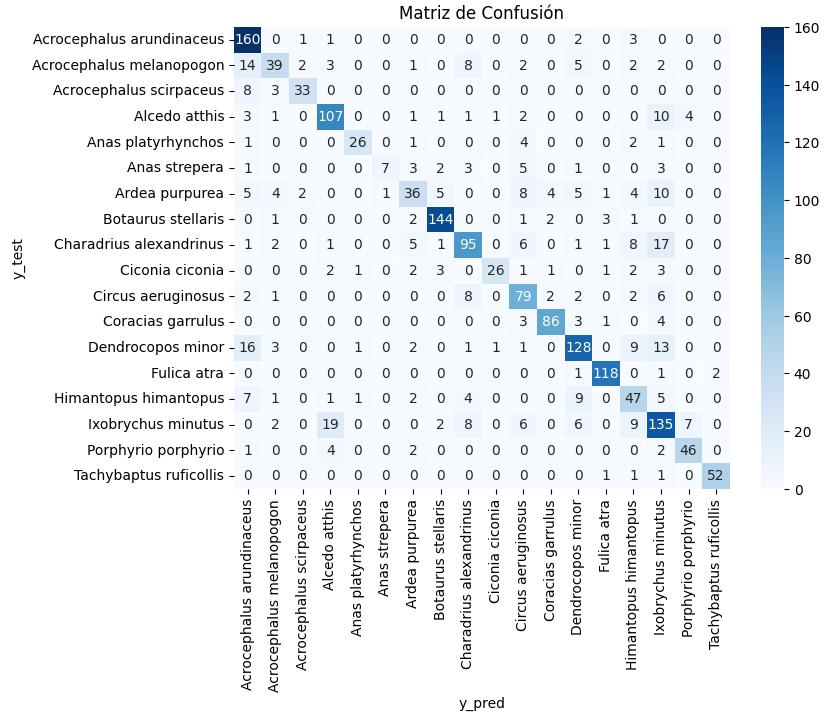
(Con train\_test\_split aleatorio) (Con train\_test\_split condicionado a audios distintos)

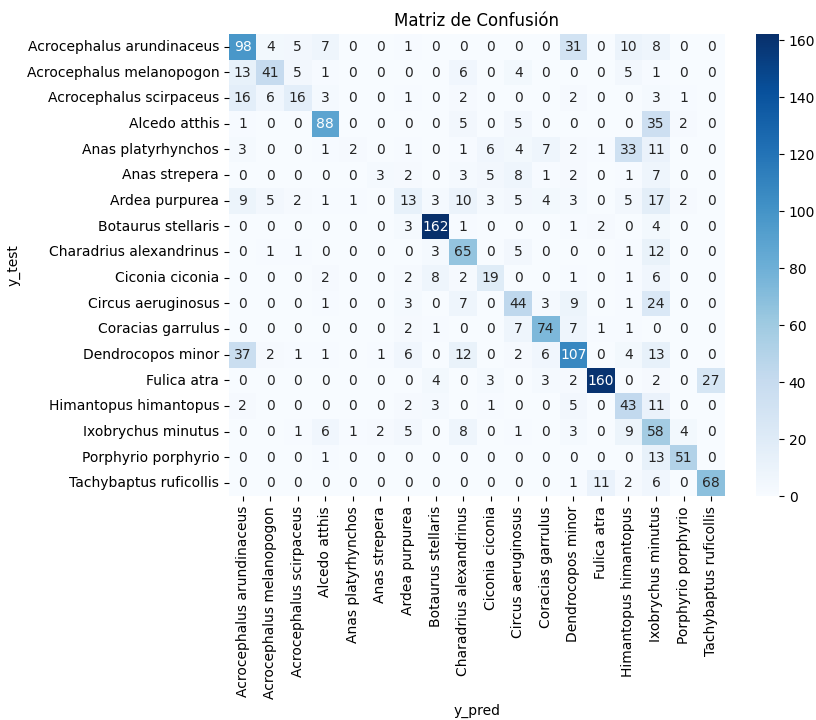
En el barrido de 1 a 100 números de ‘árboles de decisión’ evaluando el error de predicción para cada caso se observa que a mayor número de clasificadores (árboles), el error de la predicción es menor aunque el coste computacional sea mayor, es admisible por nuestros medios.

En la determinación de la importancia de las características, se sigue observando ZCR como la característica predictiva más importante seguida de la novedosa spectral\_centroid. Las demás características tienen más o menos la misma importancia, teniendo algo menor de importancia respecto el resto el amplitude\_mean, rms y spectral\_flux.

En el reporte de clasificación se observa una precisión media de 0,77 y, a excepción de 8 valores de 60 métricas, todos los demás valores superan el 60%. Se destacan las especies 20, 14, 12, 8 como especies con excelentes resultados de predicción. Se destacan las especies 7, 16, 17, 6, 2 con resultados en sus métricas más desfavorables.

Sobre la especie 6 (Anas platyrhynchos) es interesante mencionar que tiene una métrica de recall baja (0,22), una f1-score baja (0,42) pero una precisión bastante alta de 0,88. Sobre la precisión especie se puede interpretar que cuando el modelo predice esta especie es muy probable que la predicción sea correcta (pocos falsos positivos). El recall de la especie bajo nos indica que en muchas veces que la especie está presente, el modelo no la detecta y la clasifica como otra especie (muchos falsos negativos). En consecuencia a ese desbalance entre precisión y recall, la f1-score es baja. Esto concluye que es posible que la especie esté subrepresentada en los datos de entrenamiento. Puede ser una señal de que se necesitan más datos o características adicionales para mejorar la detección. Y es que justamente es la especie con menor cantidad de datos. Vemos un comportamiento en resultados similares en la especie 7 (Ardea purpurea) que es justamente la segunda especie con menor cantidad de datos.

(Con train\_test\_split aleatorio) (Con train\_test\_split condicionado a audios distintos)



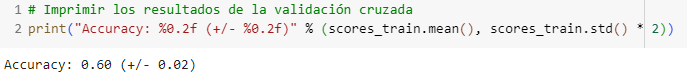
# KNN (k-Nearest Neighbors)

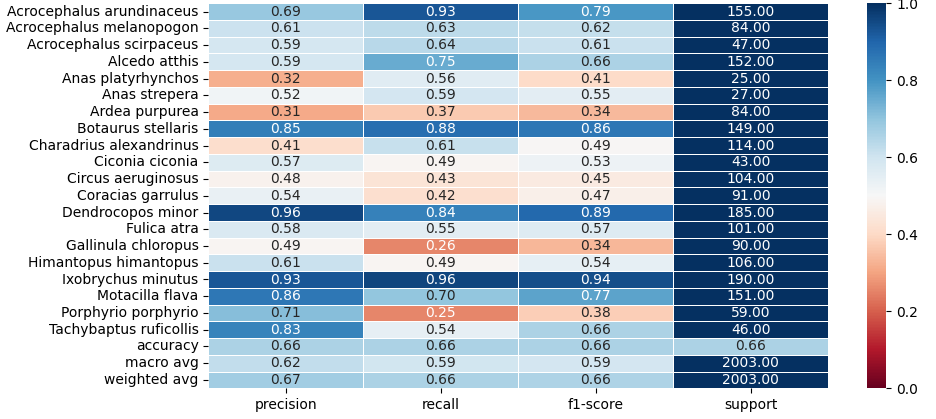
**Lectura, separación y entrenamiento de los datos**

* Lectura e imports: **code 21**
* Separación de los datos: Se usa un 65% de los datos para el entrenamiento y un 35% de los datos para el testeo. **code 22**
* Entrenamiento: Se entrenan 5 folds **code 23**

**Resultados: Precisión, Recall y F1-Score code 24**

En el modelado KNN se evalúan las mismas métricas explicadas en el apartado de RF Modelo 1.

En este caso las precisiones, sensibilidades y f1-scores obtenidas no son tan destacables. La precisión dada del modelo es de 0,60. Al ver las métricas para las distintas especies vemos que en 15 casos entre 60 los valores están por debajo del 0,50.****

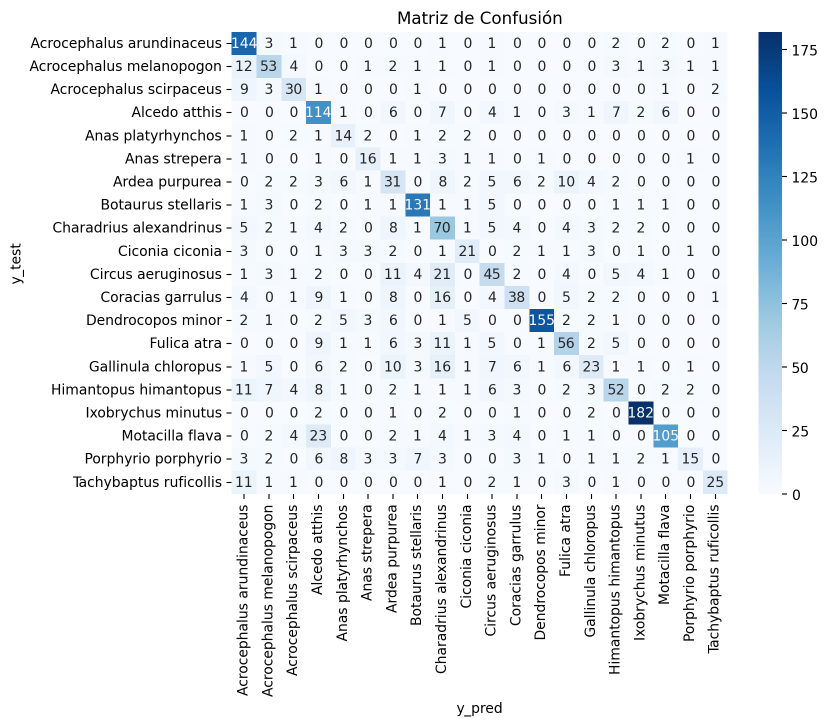
****

El modelo no se ajusta porque los datos son dispersos. Por la forma en que se distribuyen. El modelo no es adecuado para algunos pájaros como por ejemplo Anas platyrhynchos, Anas strepera, Ardea purpurea, Charadrius alexandriunus, Ciconia ciconia y Gallinula chloropus.

Con más tiempo para realizar el trabajo hubiera sido importantísimo y clave para mejorar el modelo KNN hacer un normalizado y estandarizado de datos. Porque si nos lo imaginamos ahora mismo en el modelo tenemos características con distintos rangos de valores. Por ejemplo una *feature* que va de 1000 a 2000 y otra *feature* que va de 0 a 1 entonces la primera característica estaría eclipsando la segunda característica al tener valores siempre más altos y entonces sería como si la segunda característica no se estuviera usando. Para normalizar lo que se tendría que hacer sería que los datos vayan todos de 0 a 1, con la misma media en los datos y con desviación estándar igual estandarizada. Para determinar que los datos vayan de 0 a 1 lo que se tendría que hacer es restar el máximo y dividir entre el rango (máximo menos mínimo), o sea: . Para hacer que la media esté en el 0 y la desviación estándar sea 1 se tiene que hacer la muestra (*x*) menos el valor de la media (μ) dividido entre la desviación estándar (σ), o sea la fórmula típica de la *z* de una distribución normal:

**Resultados: Matriz de confusión code 25**

La matriz de confusión compara las predicciones del modelo (y\_pred) con las especies reales de los datos de prueba (y\_test). Nos muestra la cantidad de datos que fueron precedidos en cada modo. Observando nuestra matriz observamos mayor cantidad de valores en la diagonal, hecho que corresponde a predicciones correctas. También se observan las cantidades que fueron erróneamente predichas y donde fueron mal predichas. Destacan los pájaros anteriormente citados que tienen un buen porcentaje de datos fuera de la diagonal. En concreto especies como *Circus aeruginosus*, *Dendrocopos minor* y *Fulica atra* tienen una alta cantidad de predicciones correctas. Sin embargo, hay ciertas confusiones notables. Por ejemplo, *Gallinula chloropus* es confundida varias veces con *Fulica atra*, y *Moticilla flava* con *Acrocephalus melanopogon*.

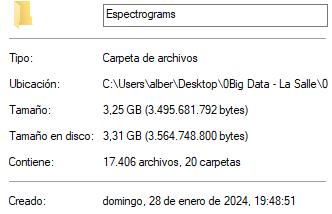


**Resultados: Precisión en función de la cantidad de vecinos code 26**

Los valores de precisión del entrenamiento y testeo son satisfactorios, 81% en el entrenamiento y 66% en el testeo. A medida que se aumenta el valor de *k* vecinos cercanos, ambos valores disminuyen. Para k=3 se logra la máxima precisión para el test.

## CNN (Convolutional Neural Networks)

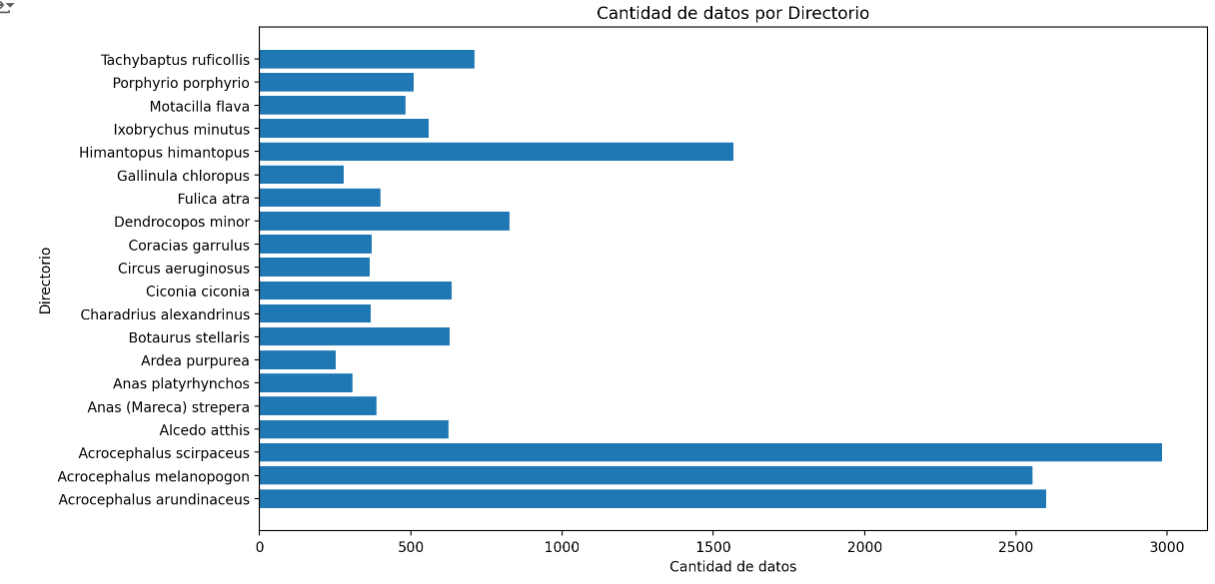
**Carga de espectrogramas e imports code 27**

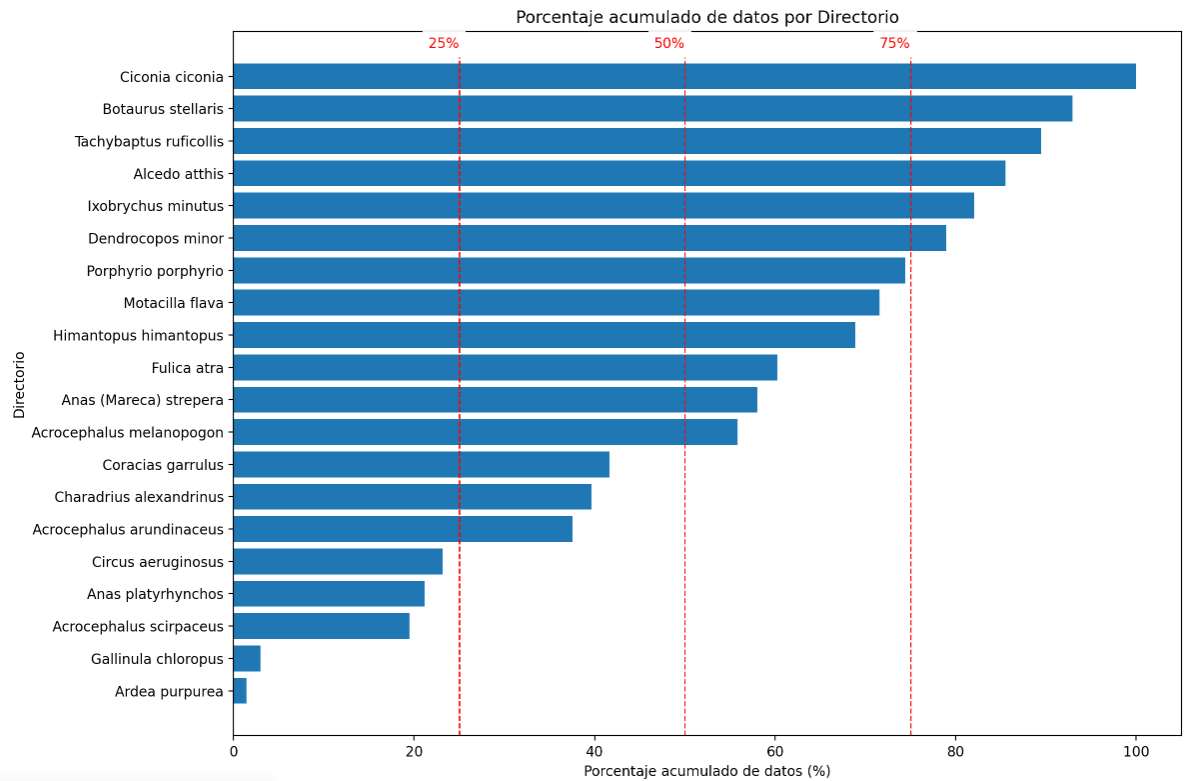
Se itera sobre las carpetas de los espectrogramas para guardar el espectrograma aplanado junto con el nombre del directorio en un dataframe.

Dado que los espectogramas tienen un tamaño de 3,31 GB y en Drive solo se pueden situar 15 GB que los tenemos llenos, creamos un nuevo correo Gmail para almacenar los espectrogramas y nuevo código para poder cargarlos.

Se itera sobre las carpetas de los espectrogramas para guardar el espectrograma junto con el nombre del directorio en un dataframe (**code 27**). Se intenta guardar el dataframe df\_espectogramas que tardó 15 min en ejecutarse en Drive en formato .parquet pero no es posible por falta de memoria RAM. Por este motivo se realiza la carga de los espectrogramas en batch de 200 espectrogramas en 200 que va guardando los dataframes de 200 datos en .parquets temporales. Finalmente los concatena todos en uno solo, borra los temporales y guarda el dataframe final “df\_espectrogramas\_final.parquet” de 4,81 GB y dimensiones 17.405 filas x 50.176 columnas en un programa que tarda 1h16min en ejecutarse. (**code 28**)

**Separación de los datos code 28**

Observamos los datos que tenemos en nuestro dataframe df\_espectrogramas\_final (**code 29**). 



Analizar todos los datos es imposible a nivel de memoria RAM. Por eso se presentan y adoptan distintas estrategias para hacerlo viable.

* Estrategia 1) Analizar los datos en cuatro etapas. Primero analizar el primer 25% de los datos (correspondiente a 5 especies de pájaros) y después los consiguientes 25% de los datos hasta realizar 4 análisis. Para esta estratégia se han ordenado (como se observa en el último gráfico) las especies para que más o menos haya el mismo número de especies en cada 25% de datos (entre 4 y 6 especies). Esta estrategia permite usar todos los datos disponibles pero supone aportar una información extra al sistema clasificador que en principio no tendría ya que en vez de tener que clasificar un espectrograma entre 20 posibles especies, ya le estamos marcando que la clasificación es entre 4-6 especies en concreto. (**code 30**)
* Estrategia 2) Analizar menos cantidad de datos para cada especie (*UnderSampling*). En el primer gráfico observamos que hay especies que tienen una gran cantidad de datos (más de 1000) mientras otras especies tienen bastante menor cantidad de datos. Al final 20 especies con tantos datos en cada especie es una cantidad enorme de información que colapsa la RAM. Para bajar esta cantidad de datos equitativamente se aplica la estrategia de rebajar la cantidad de datos de las especies con más datos fijando un máximo de datos para cada especie. Este máximo se ha fijado a 1000 datos y se ha ido rebajando hasta que la RAM del ordenador ha permitido ejecutar el programa cuando ese máximo ha sido de 250 datos. (**code 31**)
* Estrategia 3) Tras ver los perfectos resultados de las matrices de confusión de las estrategias 1 y 2, se plantea como estrategia 3 eliminar información de cada dato espectrograma ya que de cada espectrograma se cuenta con 50.175 valores que proveen mucha precisión pero en cambio ocupan memoria. Si se elimina el 70% de esa información se observa que no se pierde precisión. La eliminación de los datos se ha hecho de forma equiespaciada como muestra el código 32 con un “linspace”. Se ha hecho de esta forma por el hecho que los datos tal como están construidos tienen *overlap*. En el fichero de metadata se observa que los datos están calculados con *overlap*. Es decir, uno va de 0 a 1 segundo, el siguiente va de 0,5 a 1,5 segundos… y así. Por lo que hay una parte que los datos son iguales superpuestos. (**code 32**)

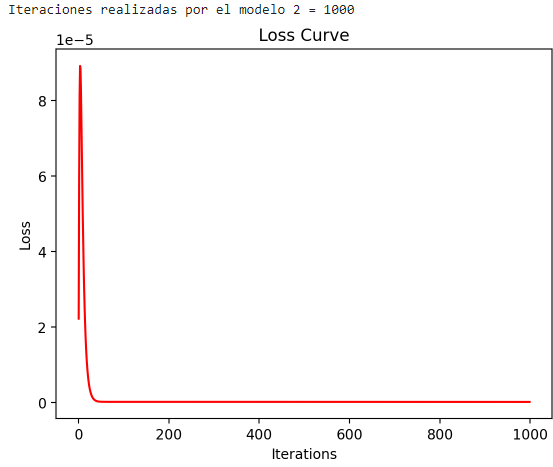
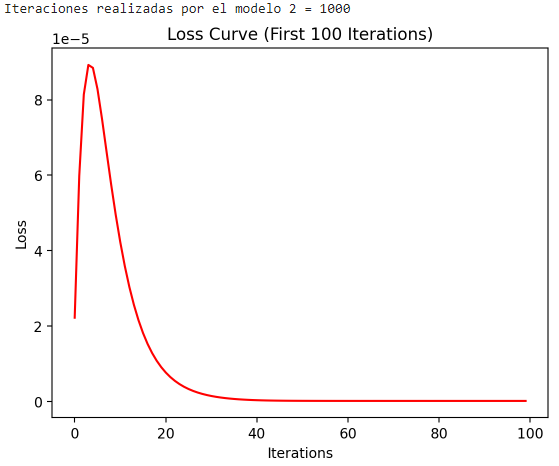
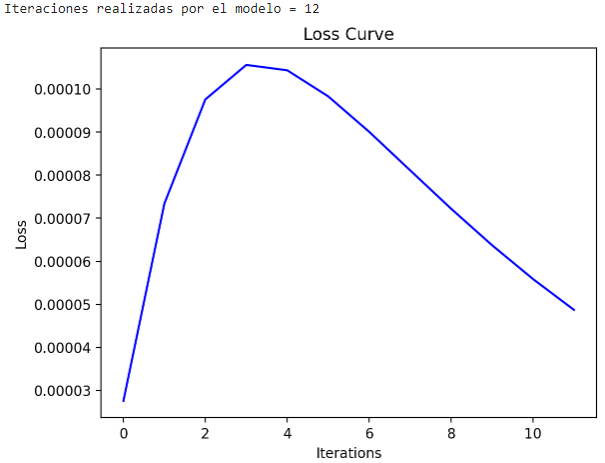
Para realizar la separación de los datos se ha partido del data frame y se han separado los datos con *X* (que no contiene el nombre del directorio a predecir) y los datos con *y* (que es el directorio). A su vez se separan los datos para entrenar (80%) y para testear (20%). Se verifica que los tamaños de los datos *X\_train* y *X\_test* sean adecuados.

**Entrenamiento code 29**

Se implementa una red neuronal artificial para la clasificación, concretamente un Multi-Layer Perceptron (MLP). Las capas de entrada reciben las características, las capas ocultas entre entrada y salida aprenden de las representaciones intermedias y en la capa de salida se produce el resultado de predicción de clasificación de la red.

Como parámetros se tienen: función de activación (activation), dimensiones de capas ocultas (hidden\_layer\_sizes), tasa de aprendizaje (learning\_rate), número máximo de iteraciones (max\_iter), algoritmo de optimización de los pesos de la red (solver)

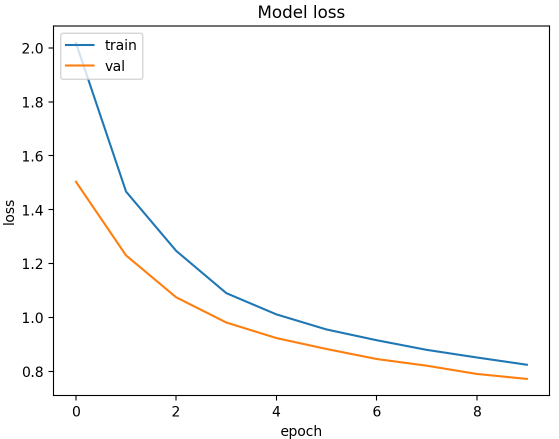
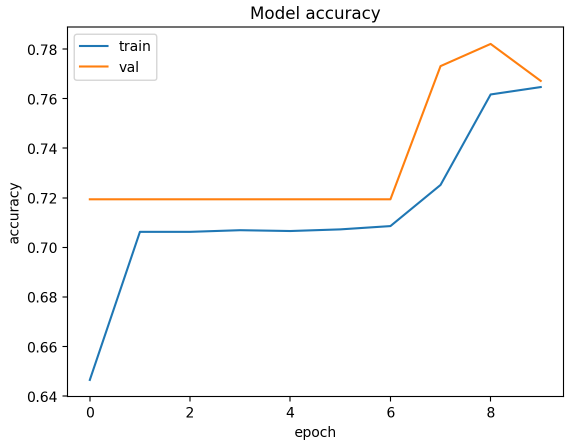
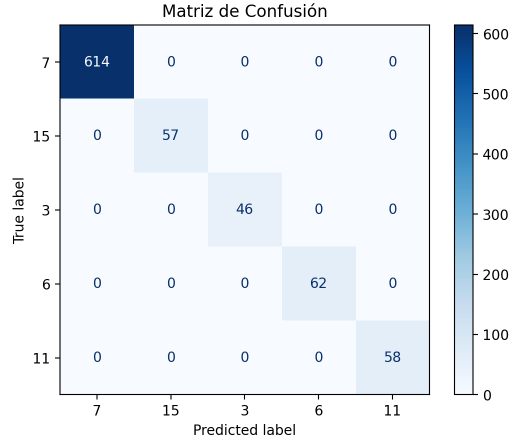
* La función de activación de las neuronas se establece “relu”, es decir solo positivo; f(x) = max(0, x).
* La cantidad de las neuronas ocultas se establece de 3 capas ocultas de 5 neuronas cada una (5, 5, 5) ya que se descubrió que esos fueron los mejores parámetros tras algunas pruebas con otro modelo distinto MPLClassifier con (20, 20) o (100, 100, 100). Sin embargo, estas dimensiones son irreales y inadecuadas por los datos que se tienen ya que en la entrada se parte de 50.000 datos (o 70% de esos en la estrategia 3) por lo que la cantidad de neuronas en la primera capa debería ser de ese orden entre 50.000 y 25.000 e ir disminuyendo la cantidad de neuronas en las capas sucesivas hasta llegar a un final de 20 neuronas que son las 20 especies a predecir. Porque por ejemplo escoger 5 neuronas de entrada supondría que los 50.000 datos de entrada están conectados a 5 neuronas lo que supondría perder mucha información. Nos podemos imaginar cómo tener una imagen de 50.000×50.000 píxeles que queda reducida a 5×5 píxeles. Visualmente en este paso se está perdiendo mucha información y no es adecuado. **code 33**
* La tasa de aprendizaje se fija constante del 1% ya que se descubrió que funcionaba mejor que el 0,01% **code 33**
* Las iteraciones primero se realizan 12 (**code 34**) y al observar que la pérdida disminuye con las iteraciones se realizan hasta 1000 (**code 35**). Observando que en las primeras 100 iteraciones la pérdida ya baja suficiente (**code 36**).
* Como solver se utiliza el método estocástico basado en el primer y segundo momento (adam).

**Resultados**

Los resultados de las distintas estrategias son los siguientes. Es sorprendente observar como las matrices de confusión son perfectas mientras que los gráficos del *model accuracy* no muestran precisiones del 100%, hecho que no cuadra. Si la matriz de confusión es perfecta la precisión debería ser del 100% y eso no cuadra. Se debería revisar el código para detectar el error. Los resultados que van apareciendo hacen pensar que el efecto de la superposición (*overlap*) de los datos. Se ha hecho de esta forma por el hecho que los datos tal como están construidos tienen *overlap*. En el fichero de metadata se observa que los datos están calculados con *overlap*. Es decir, uno va de 0 a 1 segundo, el siguiente va de 0,5 a 1,5 segundos… y así. Por lo que hay una parte en la que los datos son iguales superpuestos.

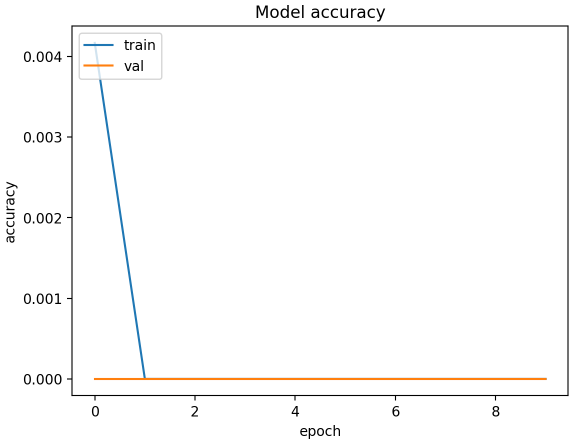
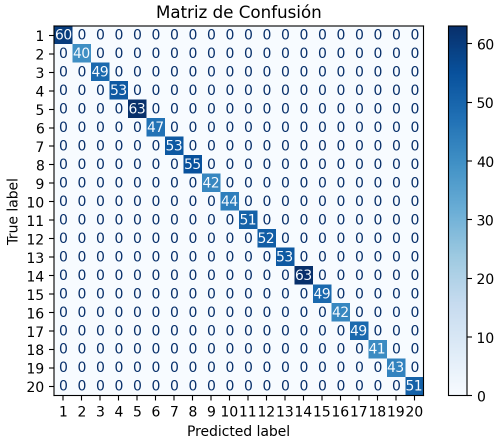
* Estrategia 1) Clasificación del 25% de las especies.

Los resultados de la matriz de confusión son perfectos.



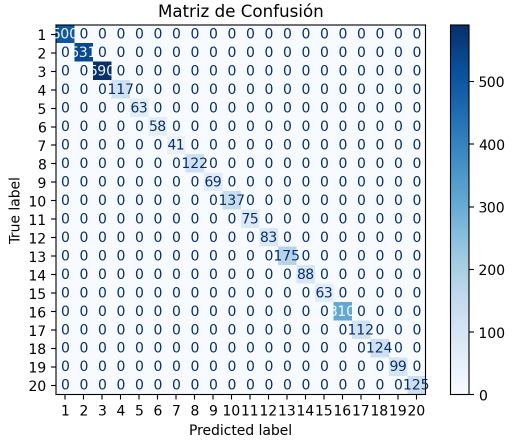
* Estrategia 2) Clasificación de un máximo de 250 datos de cada especie.

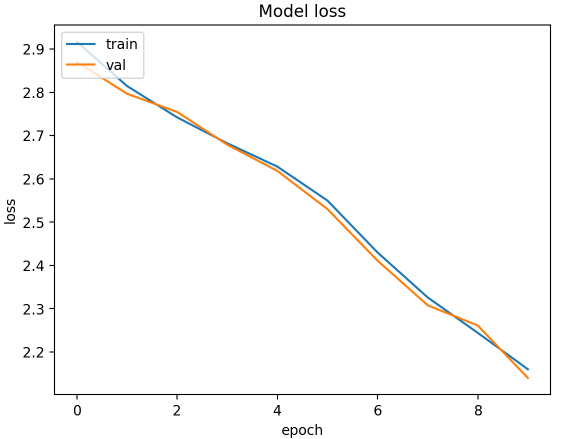
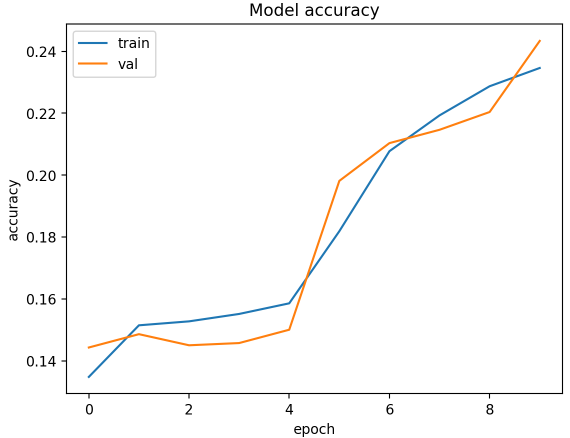
Los resultados de la matriz de confusión son perfectos.



* Estrategia 3) Clasificación de todos los datos con 70% menos de información de cada espectrograma.

Aún con un 70% menos de información, se consigue clasificar muy bien todas las especies.





Resultados obtenidos

La característica más decisiva en la predicción de las vocalizaciones de ave ha resultado ser el Zero Crossing Rate. Las características menos decisivas han sido los primeros cinco coeficientes de las deltas de primer y segundo orden. Tales resultados concuerdan con las correlaciones entre las variables, siendo el ZCR la variable que menor valor absoluto del coeficiente de regresión tiene con respecto a las otras variables y siendo los coeficientes delta las variables que tienen grandes correlaciones entre ellas.

# RF (Random Forest)

| **Modelo 1.** Directorios separados en train y test.  Zero padding para audios menores a 1 segundo. | **Modelo 2.** Mismos audios separados en train y test.  Repetición para audios menores a 1 segundo. | |
| --- | --- | --- |
| **Error de predicción vs número de árboles**  El comportamiento es similar para los dos modelos. Sin embargo, para después de 20 árboles, en el modelo 1 se llega a una asíntota horizontal del error de 0,15 mientras que para el modelo 2 es mayor de 0,4. En el modelo 1 a mayor número de árboles el sistema se comporta para disminuir claramente el error. En el modelo 2 también sucede lo mismo pero con algunas variaciones algo más bruscas de subida y bajada del error entorno a movimientos de 1-2 árboles. Sobretodo en la zona alrededor de los 10 árboles. | | |
|  |  | |
| **Reporte de clasificación (precisión, recuperación y f1-score)**  El modelo 1 tiene valores de precisión, recuperación y f1-score promedios considerablemente más altos que el modelo 2, lo que indica que el modelo 1 es más efectivo en la clasificación de las especies. Mientras el modelo 1 muestra un rendimiento muy bueno y uniforme en las especies, el modelo 2 tiene un rendimiento mucho más variable y tiene serios problemas en reconocer ciertas especies. | | |
|  |  | |
| **Matriz de confusión**  En general en modelo 1 la mayoría de las predicciones son verdaderas ya que muestra una diagonal principal bien definida. En comparación, en el modelo 2 la diagonal principal está menos definida y tiene más dispersiones fuera de la diagonal; las confusiones son más frecuentes. | | |
|  | |  |

# 

# KNN (k-Nearest Neighbors)

| **KNN.** Separación aleatoria de los datos 65% train, 35% test. Entrenamiento con 5 folds. | **RF Modelo 2.** Mismos audios separados en train y test.  Repetición para audios menores a 1 segundo. |
| --- | --- |
| **Reporte de clasificación (precisión, recuperación y f1-score)**  El modelo KNN parece ser más equilibrado en general, con menos casos de muy bajo *recall* comparado con RF. Sin embargo, el modelo KNN sigue teniendo dificultades para distinguir ciertas especies como lo indican las bajas métricas de recall y f1-score. Por lo que respecta al modelo RF, aunque presenta buena precisión en algunas clases (14, 8 y 19), destaca por su poca recuperación en ciertas clases donde tiene predilección por clasificar incorrectamente los audios de las especies 5, 6 y 7. | |
|  |  |
| **Matriz de confusión**  En general en modelo kNN la mayoría de las predicciones son verdaderas ya que muestra una diagonal principal bien definida. En comparación, en el modelo 2 la diagonal principal está menos definida y tiene más dispersiones fuera de la diagonal; las confusiones son más frecuentes. | |
|  |  |

CNN (Convolutional Neural Networks)

| Estrategia 1) Clasificación del 25% de las especies. | Estrategia 2) Clasificación de un máximo de 250 datos de cada especie. | Estrategia 3) Clasificación de todos los datos con 70% menos de información de cada espectrograma. |
| --- | --- | --- |
| La clasificación de las especies es perfecta. | La clasificación de las especies es perfecta. | La clasificación de las especies es perfecta. |
| Con una precisión constante de alrededor de 70-72% en las primeras seis épocas y una mejora del 6%, esta estrategia demuestra cierta consistencia en el reconocimiento de especies. | Debido a la poca cantidad de datos, la precisión del modelo por épocas no aporta información significativa. | Tiene la precisión más baja (0,14) y solo mejora un 10% después de varias épocas. Este bajo rendimiento indica que la reducción del 70% en la información ha afectado negativamente la capacidad del modelo para distinguir especies. |

Conclusiones

Como conclusiones generales del trabajo, se determinan **factores que influyeron en el rendimiento** de los modelos:

* Cantidad y distribución de los datos de entrenamiento: La cantidad insuficiente de datos afectó significativamente el rendimiento de los modelos. Especies subrepresentadas en los datos de entrenamiento, como Anas platyrhynchos, concluyeron con métricas de recall bajas.
* Diversidad en la duración de los datos: El amplio rango de duración temporal de las vocalizaciones de las aves dificultó la extracción de las características acústicas de forma satisfactoria y facilitó la confusión en comparar vocalizaciones similares.
* Complejidad de las características acústicas: Las características acústicas complejas y la dificultad para extraerlas con exactitud y extraerlas satisfactoriamente dificultaron la clasificación precisa entre los cantos de algunas especies similares. Especies con vocalizaciones similares presentaron mayores tasas de confusión.
* Estrategias de Preprocesamiento: La efectividad de las estrategias de preprocesamiento, como el zero padding y la repetición de audios cortos, influyó en el rendimiento de los modelos. El zero padding resultó ser más efectivo en el caso de RF Modelo 1.

Como **línea de trabajo futura** que continúe en la mejora de los modelos y del proceso de clasificación el presente trabajo se proclama que aunque cada modelo tiene sus propias fortalezas y debilidades, hay margen de mejora. El enfoque debe estar en mejorar cantidad y distribución de los datos de entrenamiento, optimizar la extracción de características e indagar en los métodos avanzados de procesamiento de señales para discernir diferencias sutiles en los cantos de los pájaros:

* Mejorar la cantidad y distribución de los datos: Recolectar más datos de alta calidad y sobre todo asegurar una representación equilibrada de todas las especies mejorará el rendimiento de los modelos. Se puede plantear utilizar técnicas *Upsampling* que evitaría tener que determinar manualmente los inicios y finales de los cantos.
* Optimización de los parámetros: Ajustar los parámetros de cada modelo puede mejorar el equilibrio entre precisión y recall. Realizar una búsqueda más exhaustiva de parámetros para encontrar la configuración óptima.
* Mejorar el preprocesamiento de datos: Aplicar técnicas avanzadas de preprocesamiento y extracción de características para capturar mejor las sutilezas de las vocalizaciones de las aves. Se puede considerar el uso de técnicas de filtrado y normalización.
* Uso de técnicas ensemble: Combinar las salidas de múltiples modelos puede ser una línea de trabajo futura que permita proporcionar una clasificación más precisa y generalizada al considerar diferentes características de los datos.

Como **conclusiones** del trabajo **específicas para cada modelo** se determinan las siguientes:

# RF (Random Forest)

El modelo 1 mostró un rendimiento superior con una precisión media de 0.77 con lo que generaliza mejor a los datos de prueba que el modelo 2. La matriz de confusión indicó menos errores y una mayor precisión en la predicción de las especies 20, 14, 12 y 8 que destacaron por sus excelentes resultados de predicción. En cambio el modelo 2 presentó un rendimiento más inconsistente con una mayor cantidad de confusiones. Los resultados de recuperación y f1-score indicaron grandes confusiones en la detección de las especies 5, 6, 7, 17.

La estrategia en el tratamiento de audios cortos a través de repetición (modelo 2) resultó no ser tan efectiva como el zero padding utilizado en el modelo 1. La precisión fue más baja y la matriz de confusión mostró más errores. Aparentemente, separar los directorios para el entrenamiento y la prueba en el modelo 1 parece proporcionar una ventaja significativa en comparación con el uso de los mismos audios para cada fase train/test en el modelo 2. Esto podría deberse a una mejor representación y diversidad de los datos en el conjunto de prueba en el Modelo 1.

# KNN (k-Nearest Neighbors)

A nivel de rendimiento general, el modelo KNN tuvo una precisión media de 0.60. Aunque equilibrado en general, mostró dificultades para distinguir ciertas especies, con métricas de recall y F1-score bajas en algunos casos. Las especies como Anas platyrhynchos, Anas strepera, Ardea purpurea, Charadrius alexandrinus, Ciconia ciconia y Gallinula chloropus fueron particularmente problemáticas.

La razón principal por la cual el modelo no es satisfactorio es por la falta de normalización y estandarización como bien se ha discutido y detallado ya en los resultados del modelo. Por lo tanto como línea de trabajo futura se determina normalizar y estandarizar los datos previo a modelo KNN: Es clave realizar la normalización y estandarización de los datos antes de realizar el modelo KNN. Es una línea de trabajo futura poco costosa a nivel de cantidad temporal de trabajo y que daría muchos mejores resultados de predicción. En el apartado de resultados del KNN se describe como hacer el modelado. Es tan simple como normalizar los datos de 0 a 1; y estandarizar la media y desviación estándar . Sería interesante evaluar la mejora que eso supondría respecto al presente trabajo hecho sin la normalización.

Otra razón por la cual el modelo no se acaba de ajustar correctamente a los datos es debido a la forma en cómo se distribuyen ya que, por su naturaleza, los datos son dispersos. Este hecho implica que el modelo clasificador por vecino más cercano quizás no sea el mejor modelo para el caso.

CNN (Convolutional Neural Networks)

El método clasificador de los espectrogramas de las vocalizaciones por redes neuronales concluyó con resultados de clasificación muy buenos sobre todo en la matriz de confusión que fueron mal programados. A pesar de los resultados, se concluye aspectos sobre los factores que influyeron en el rendimiento del modelo y en la posibilidad por memoria RAM de ejecutar el modelo.

* Aplicar la estrategia de clasificar las especies en cuatro etapas cada una conteniendo el 25% de las especies a la vez que el 25% no fue tarea fácil por el hecho de tener que cuadrar esas dos métricas. En la misma línea, la RAM no fue suficiente para procesar la siguiente etapa de datos. El modelo mostró una precisión constante de alrededor del 70-72% en las primeras seis épocas, con una mejora del 6%. La estrategia demostró cierta consistencia en el reconocimiento de especies. Sin embargo, esta estrategia introduce información al modelo que en principio no se dispondría: acotar en cada etapa las especies posibles a las cuales se puede clasificar el audio.
* Clasificar un máximo de 250 datos de cada especie tuvo la precisión más baja (0,14) y solo mejoró un 10% después de varias épocas. La poca cantidad de datos afectó negativamente el rendimiento al modelo. Esta estrategia podría generar un sobreajuste de los datos.
* Estrategia 3: Clasificación de todos los datos con 70% menos de información de cada espectrograma resultó en un bajo rendimiento, indicando que la reducción de información afectó la capacidad del modelo para distinguir especies. La estrategia puede conducir a un aprendizaje insuficiente.

Para trabajos futuros se exploran las siguientes recomendaciones:

* Aumentar los datos usados y la cantidad RAM: Claramente, aumentar el uso de toda la información disponible con todos los datos o incluso aumentar más los datos, ayudaría a mejorar el rendimiento del modelo y evitar el sobreajuste. Hay que recordar que los modelos neuronales funcionan muy bien con grandes volúmenes de datos. Sin embargo, para lograr eso, hay que considerar obtener más capacidad RAM para ser capaz de procesar toda esa información.
* Cambiar la estrategia de uso de la RAM: En vez de intentar cargar todos los datos en en RAM trabajando con datasets muy grandes del orden de Big Data, lo interesante sería ir haciendo las lecturas y el pasado de los ficheros a medida que se va realizando el entrenamiento. O sea, hacer por grupos de datos la lectura y entrenamiento del modelo con bucles. De forma automática se podría usar un *DataLoader*.
* Mejorar selección de las características: En la estrategia 3, utilizar técnicas más avanzadas para seleccionar y preservar la información más relevante de los espectrogramas ayudaría gratamente al modelo a realizar un entrenamiento más satisfactorio sin tener un sobreuso computacional.
* Cambiar la arquitectura de datos del modelo a uno adecuado. En vez de programar la arquitectura con 3 capas de 5 neuronas lo que sería recomendable sería trabajar con una capa inicial del orden de la dimensión de la entrada de los datos (50.000) y progresivamente ir reduciendo la cantidad de neuronas en cada capa hasta llegar a la última capa con 20 neuronas correspondientes a las 20 especies a predecir.

Anexo código

**code 1**

| # Cargar el archivo de audio MP3  mp3\_path = '/content/drive/MyDrive/BigData/DataMining/Practica1/Audiofiles/Acrocephalus arundinaceus/XC417157.mp3'  y, sr = librosa.load(mp3\_path)  # Mostrar la forma de onda del audio  plt.figure(figsize=(14, 5))  librosa.display.waveshow(y, sr=sr)  plt.title('Forma de Onda del Audio')  plt.show()  # Escuchar el audio  ipd.Audio(y, rate=sr) |
| --- |

**code 2**

Se lee un audio de “Audiofiles” y se guarda según los cortes de tiempo en “Audio\_Clip”. De esta forma, la carga de computación se reparte con el tiempo. Porque el programa es capaz de ir leyendo uno a uno los archivos e irlos guardando otra vez con los cortes de tiempo hechos. Este programa tarda bastante tiempo en ejecutarse. Por eso es buena opción realizarlo de entrada una vez y ya guardar el resultado en vez de hacerlo cada vez que se quiere realizar un análisis. Así ya se tienen los archivos de audio guardados y después solo se tendrá que volver a leer los archivos .wav que se han ido guardando.

| input\_folder = r'/content/drive/MyDrive/BigData/DataMining/Practica1/Audiofiles/'  output\_folder = r'/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip'  def trim\_and\_save\_audio(input\_folder, output\_folder):  for bird\_folder in os.listdir(input\_folder):  bird\_folder\_path = os.path.join(input\_folder, bird\_folder)  print(bird\_folder)  if os.path.isdir(bird\_folder\_path):  output\_bird\_folder\_path = os.path.join(output\_folder, bird\_folder)  os.makedirs(output\_bird\_folder\_path, exist\_ok=True)  for audio\_file in os.listdir(bird\_folder\_path):  if audio\_file.endswith(".mp3"):  audio\_path = os.path.join(bird\_folder\_path, audio\_file)  text\_file\_path = os.path.join(bird\_folder\_path, audio\_file.replace(".mp3", ".txt"))  if os.path.exists(text\_file\_path):  with open(text\_file\_path, 'r') as text\_file:  lines = text\_file.readlines()  for i, line in enumerate(lines, start=1):  start\_time, end\_time, \_ = line.strip().split('\t')  start\_time, end\_time = float(start\_time), float(end\_time)  print(f"Start: {start\_time}, end: {end\_time}")  # Load audio with librosa  y, sr = librosa.load(audio\_path, sr=None)  # Trim audio  start\_frame = int(start\_time \* sr)  end\_frame = int(end\_time \* sr)  trimmed\_audio = y[start\_frame:end\_frame]  # Save trimmed audio  output\_audio\_name = f"{os.path.splitext(audio\_file)[0]}\_{i}.wav"  output\_audio\_path = os.path.join(output\_bird\_folder\_path, output\_audio\_name)  sf.write(output\_audio\_path, trimmed\_audio, sr)  print(f"Trimmed audio saved: {output\_audio\_path}")  trim\_and\_save\_audio(input\_folder, output\_folder) |
| --- |

**code 3**

| FRAME\_SIZE = 1024 # Muestras que tiene la ventana de audio analizada.  HOP\_LENGTH = 512 # Muestras que se santan de ventana en ventana.  N\_MFCC = 13 # Número de MFCC  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(np.sum(signal[i:i+frame\_size]\*\*2) / frame\_size)  rms.append(rms\_current\_frame) # media de la ventana actual (añade cada una)  return np.mean(rms) # media entre todas las ventanas  def calcular\_caracteristicas(y, sr, FRAME\_SIZE, HOP\_LENGTH, audio\_file):  # Calcular las características del audio  mfcc = librosa.feature.mfcc(y=y, n\_mfcc=13, sr=sr)  mfcc\_scaled = np.mean(mfcc, axis=1)  delta1\_mfcc = librosa.feature.delta(mfcc\_scaled, mode='nearest')  delta2\_mfcc = librosa.feature.delta(mfcc\_scaled, mode='nearest', order=2)  zcr = librosa.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)  features = np.concatenate(  (zcr, rms, spectral\_centroid, spectral\_flux, spectral\_bandwidth, mfcc.T, delta1\_mfcc, delta2\_mfcc), axis=1  )  # Crear una lista para almacenar las filas de características  data = []  for feature in features:  row = feature.tolist()  row.append(audio\_file) # Agregar el nombre del archivo de audio  data.append(row)  return np.array(data) # Devolver como un array de numpy |
| --- |

**demostración 3**

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

| Forma 1    Forma 2:   * Energía potencial elástica de un movimiento armónico simple: * Fuerza elástica de tensión: Donde constante elástica, longitud del resorte * Tensión de una cuerda tensa: Donde velocidad de propagación, y la densidad lineal del medio:   Con todo, sustituyendo expresiones de arriba abajo:  Para la potencia (energía por unidad de tiempo):  Para la intensidad (potencia por unidad de espacio del frente de onda, en este caso tridimensional): |
| --- |

**code 4**

| audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  max\_length = 1 # Segundos que van a tener todos los audios  def extract\_audio\_features(audio\_folder):  audio\_features\_data = []  # Iterar sobre las carpetas de las especies  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  # Iterar sobre los archivos de audio dentro de cada carpeta de especies  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  if len(y) < max\_length\*sr: # Si el audio es más pequeño que 1 segundo, hace zero padding  expanded\_y = np.zeros(max\_length \* sr, dtype=y.dtype)  expanded\_y[:len(y)] = y  y = expanded\_y  # Calcular los MFCC  mfcc = librosa.feature.mfcc(y=y, n\_mfcc=N\_MFCC, sr=sr)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  audio\_features\_data.append(fila)  else: # Si el audio es más grande que 1 segundo, corta el audio a pedazos de 1 segundo.  num\_segments = len(y) // (max\_length \* sr) # Número de segmentos que se pueden obtener  for i in range(num\_segments):  segment = y[i \* max\_length \* sr: (i + 1) \* max\_length \* sr] # Corte del segmento  # Calcular los MFCC  mfcc = librosa.feature.mfcc(y=segment, n\_mfcc=N\_MFCC, sr=sr)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  audio\_features\_data.append(fila)  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  columns = [f'mfcc\_{i + 1}' for i in range(N\_MFCC)] + ['label']  df = pd.DataFrame(audio\_features\_data,columns=columns)  return df  df = extract\_audio\_features(audio\_folder)  # Save df\_mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_mfcc.parquet"), index=False) |
| --- |

**code 5**

| audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  max\_length = 1 # Segundos que van a tener todos los audios  def extract\_audio\_features(audio\_folder):  audio\_features\_data = []  audio\_file="XC520116\_3.wav"  audio\_path="/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/Tachybaptus ruficollis/XC520116\_3.wav"  try:  y, sr = librosa.load(audio\_path)  print (len(y), len(y) // (max\_length \* sr))  #if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  if len(y) < max\_length\*sr: # Si el audio es más pequeño que 1 segundo, hace zero padding  expanded\_y = np.zeros(max\_length \* sr, dtype=y.dtype)  expanded\_y[:len(y)] = y  y = expanded\_y  # Calcular los MFCC  mfcc = librosa.feature.mfcc(y=y, n\_mfcc=13, sr=sr)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  audio\_features\_data.append(fila)  else: # Si el audio es más grande que 1 segundo, corta el audio a pedazos de 1 segundo.  num\_segments = len(y) // (max\_length \* sr) # Número de segmentos que se pueden obtener  for i in range(num\_segments):  segment = y[i \* max\_length \* sr: (i + 1) \* max\_length \* sr] # Corte del segmento  # Calcular los MFCC  mfcc = librosa.feature.mfcc(y=segment, n\_mfcc=13, sr=sr)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  audio\_features\_data.append(fila)  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  columns = [f'mfcc\_{i + 1}' for i in range(N\_MFCC)] + ['label']  df = pd.DataFrame(audio\_features\_data,columns=columns)  return df  df = extract\_audio\_features(audio\_folder)  df.tail()  # Save df\_mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_mfcc.parquet"), index=False) |
| --- |

**code 6**

| audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  max\_length = 1 # Segundos que van a tener todos los audios  def extract\_audio\_features(audio\_folder):  df = pd.DataFrame()  audio\_file="XC520116\_3.wav"  audio\_path="/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/Tachybaptus ruficollis/XC520116\_3.wav"  try:  y, sr = librosa.load(audio\_path)  print (len(y), len(y) // (max\_length \* sr))  #if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  if len(y) < max\_length\*sr: # Si el audio es más pequeño que 1 segundo, hace zero padding  # Calcular las caracteristicas  mfcc = librosa.feature.mfcc(y=y, n\_mfcc=13, sr=sr)  frame\_size = len(segment) // 4  hop\_length = frame\_size // 4  #caracteristicas = calcular\_caracteristicas (y,sr,frame\_size,hop\_length)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  df = pd.concat([df, pd.DataFrame([fila])], ignore\_index=True)  else: # Si el audio es más grande que 1 segundo, corta el audio a pedazos de 1 segundo.  num\_segments = len(y) // (max\_length \* sr) # Número de segmentos que se pueden obtener  for i in range(num\_segments):  segment = y[i \* max\_length \* sr: (i + 1) \* max\_length \* sr] # Corte del segmento  # Calcular las caracteristicas  mfcc = librosa.feature.mfcc(y=segment, n\_mfcc=13, sr=sr)  frame\_size = len(segment) // 4  hop\_length = frame\_size // 4  #caracteristicas=calcular\_caracteristicas (y,sr,frame\_size,hop\_length)  for feature in mfcc.T:  fila = feature.tolist() + [audio\_file]  df = pd.concat([df, pd.DataFrame([fila])], ignore\_index=True)  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  df.columns = [f'mfcc\_{i + 1}' for i in range(13)] + ['label']  return df  df = extract\_audio\_features(audio\_folder)  df.tail()  # Save df\_mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_mfcc.parquet"), index=False) |
| --- |

**code 7**

| audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  max\_length = 1 # Segundos que van a tener todos los audios  def extract\_audio\_features(audio\_folder):  audio\_features\_data = []  # Iterar sobre las carpetas de las especies  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  # Iterar sobre los archivos de audio dentro de cada carpeta de especies  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  if len(y) < max\_length\*sr: # Si el audio es más pequeño que 1 segundo, hace zero padding  expanded\_y = np.zeros(max\_length \* sr, dtype=y.dtype)  expanded\_y[:len(y)] = y  y = expanded\_y  # Calcular las características del audio  audio\_feature\_dict = calcular\_caracteristicas(y, sr, FRAME\_SIZE, HOP\_LENGTH, audio\_file)  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  continue  audio\_features\_data.append(audio\_feature\_dict)  df = pd.DataFrame(audio\_features\_data)  return df  df = extract\_audio\_features(audio\_folder) |
| --- |

**code 8**

| def extract\_audio\_features(audio\_folder):  audio\_features\_data = []  # Iterar sobre las carpetas de las especies  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  # Iterar sobre los archivos de audio dentro de cada carpeta de especies  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE \* 2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  if len(y) < max\_length \* sr: # Si el audio es más pequeño que 1 segundo, hace zero padding  expanded\_y = np.zeros(max\_length \* sr, dtype=y.dtype)  expanded\_y[:len(y)] = y  y = expanded\_y  # Calcular las características del audio  audio\_feature\_dict = calcular\_caracteristicas(y, sr, FRAME\_SIZE, HOP\_LENGTH, audio\_file) # Pasar audio\_file como argumento  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  continue  audio\_features\_data.append(audio\_feature\_dict)  df = pd.DataFrame(audio\_features\_data)  return df  df = extract\_audio\_features(audio\_folder)  df.tail() |
| --- |

**code 9**

| # Sanity check para ver que las longitudes son iguales en todos los casos  df.delta1.apply(len).value\_counts()  # Sanity check para ver que las longitudes son iguales en todos los casos  df.delta1.apply(len).value\_counts()  ## Separar las listas delta1 y delta 2.  # Función que convierte cada lista en una Serie de pandas que expande en múltiples columnas  df\_delta1\_expanded = df['delta1'].apply(pd.Series)  df\_delta2\_expanded = df['delta2'].apply(pd.Series)  # Nombre de las columnas  df\_delta1\_expanded.columns = [f'delta1\_{i}' for i in range(len(df\_delta1\_expanded.columns))]  df\_delta2\_expanded.columns = [f'delta2\_{i}' for i in range(len(df\_delta2\_expanded.columns))]  # Concatenar  df\_result = pd.concat([df, df\_delta1\_expanded], axis=1)  df\_result = pd.concat([df\_result, df\_delta2\_expanded], axis=1)  df\_delta = df\_result  # Eliminar las columnas originales  df\_delta = df\_delta.drop(columns=['delta1', 'delta2'])  df\_delta.tail()  # Save df\_delta  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df\_delta.to\_parquet(os.path.join(main\_folder, "df\_delta.parquet"), index=False) |
| --- |

**code 10**

| # Leer delta, sin mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df\_delta = pd.read\_parquet(os.path.join(main\_folder, "df\_delta.parquet"))  df\_delta.tail(2) |
| --- |

**code 11**

| audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  data = []  FRAME\_SIZE = 1024  # Dataframe con el nombre de la especie (directorio) y del archivo de audio  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  data.append((species\_folder, audio\_file))  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  df\_names = pd.DataFrame(data, columns=['Directorio', 'NombreArchivo'])  df\_names.tail(2) |
| --- |

**code 12**

| # Eliminar la terminación ".wav"  df\_names['NombreArchivo'] = df\_names['NombreArchivo'].str.replace('.wav', '')  df = pd.concat([df\_names, df\_delta], axis=1)  df.tail(2)  # Save Data Frame usado en el Machine Learning  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_ml.parquet"), index=False) |
| --- |

**code 13**

| import seaborn as sns  sns.countplot(data=y\_train, palette="Set3")  plt.xlabel("y train count", fontsize=10)  plt.show()  # Gráfica de las features  number\_of\_features = list(range(X\_train.shape[1]))  print("Feature importances Random forest:", RF.feature\_importances\_)  plt.barh(number\_of\_features, RF.feature\_importances\_)  plt.yticks(number\_of\_features, X.columns)  plt.xlabel("Feature Importances")  plt.ylabel("Features")  plt.show() |
| --- |

**code 14**

| # Leer delta, sin mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df\_delta = pd.read\_parquet(os.path.join(main\_folder, "df\_delta.parquet"))  df\_delta.tail(2)  plt.figure(figsize=(10,9))  sns.heatmap(df\_delta.corr()) |
| --- |

**code 15**

| # Leer datos  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df = pd.read\_parquet(os.path.join(main\_folder, "df\_ml.parquet"))  df.tail(2)  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import classification\_report  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score  from sklearn.metrics import confusion\_matrix  from sklearn.ensemble import RandomForestClassifier  import warnings  warnings.filterwarnings(action="default") |
| --- |

**code 16**

| # Extraer el nombre base antes de la \_ en 'NombreArchivo'  # Este nombre es el identificador único del audio. Después hay \_número indicando el número de corte  df['NombreBase'] = df['NombreArchivo'].str.split('\_').str[0]  # Separar X e y  X = df.drop(columns=['Directorio', 'NombreArchivo', 'NombreBase'])  y = df['Directorio']  grouped = df.groupby('NombreBase')  train\_indices = []  test\_indices = []  # Separar por nombres base manteniendo la proporción 35% datos de test  for \_, group in grouped:  if len(test\_indices) < 0.35 \* len(df):  test\_indices.extend(group.index)  else:  train\_indices.extend(group.index)    train\_indices = list(set(train\_indices))  test\_indices = list(set(test\_indices))  # Obtener X\_train, X\_test, y\_train, y\_test  X\_train, X\_test = X.loc[train\_indices], X.loc[test\_indices]  y\_train, y\_test = y.loc[train\_indices], y.loc[test\_indices]  X\_train = X\_train.values  y\_train = y\_train.values  import seaborn as sns  sns.countplot(data=y\_train, palette="Set3")  plt.xlabel("y train count", fontsize=10)  plt.show() |
| --- |

**code 17**

| # Entrenar el modelo de Random Forest con diferentes números de árboles y calcular el error de predicción  n\_trees = 100  errors = []  for i in range(1, n\_trees + 1):  rf = RandomForestClassifier(n\_estimators=i, random\_state=42)  rf.fit(X\_train, y\_train)  error = 1 - rf.score(X\_test, y\_test)  errors.append(error)  # Graficar el error de predicción en función del número de árboles  plt.figure(figsize=(10, 6))  plt.plot(range(1, n\_trees + 1), errors, marker='o', linestyle='-')  plt.xlabel('Número de árboles')  plt.ylabel('Error de predicción')  plt.title('Error de predicción del Random Forest')  plt.grid(True)  plt.show()  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 18**

| # Gráfica de las features  number\_of\_features = list(range(X\_train.shape[1]))  print("Feature importances Random forest:", RF.feature\_importances\_)  plt.barh(number\_of\_features, RF.feature\_importances\_)  plt.yticks(number\_of\_features, X.columns)  plt.xlabel("Feature Importances")  plt.ylabel("Features")  plt.show()  # Accuracy  print("Accuracy - Datos de train: {} " .format(accuracy\_score(y\_train, y\_train\_pred)))  print("Accuracy - Datos de test: {} ". format(accuracy\_score(y\_test, y\_test\_pred)))  from sklearn.metrics import classification\_report  import seaborn as sns  import matplotlib.pyplot as plt  # 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()  visual\_classification\_report(y\_test, y\_test\_pred) |
| --- |

**code 19**

| # Matriz de confusión  conf\_matrix = confusion\_matrix(y\_test, y\_test\_pred)  classes = np.unique(y\_test)  plt.figure(figsize=(8, 6))  sns.heatmap(conf\_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=classes, yticklabels=classes)  plt.xlabel('y\_pred')  plt.ylabel('y\_test')  plt.title('Matriz de Confusión')  plt.show() |
| --- |

**code 20**

| # Gráfica de los árboles  from sklearn import tree  fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (40,8))  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 = 16)  ###ESTE CÓDIGO SE REPITE PARA HACER LOS DIAGRAMAS ÁRBOLES MÁS GRANDES Y LEIBLES |
| --- |

**code 21**

| # Leer datos  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df = pd.read\_parquet(os.path.join(main\_folder, "df\_ml.parquet"))  df.tail(2)  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.metrics import classification\_report  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import accuracy\_score  import matplotlib.pyplot as plt  from sklearn.model\_selection import cross\_val\_score |
| --- |

**code 22**

| # Separamos el dataframe según los datos X e y. El target es adivinar el nombre de la especie, o sea el directorio.  X = df.drop(columns=["Directorio"])  X = X.drop(columns=["NombreArchivo"])  y = df["Directorio"]  # Separamos los datos en un 65% para entrenar el modelo y en un 35% para su testeo  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35, random\_state=10) |
| --- |

**code 23**

| KNN = KNeighborsClassifier(n\_neighbors=3)  KNN.fit(X\_train, y\_train)  # Aplicar validación cruzada con 5 folds  scores\_train = cross\_val\_score(KNN, X\_train, y\_train, cv=5)  y\_train\_pred = KNN.predict(X\_train)  y\_test\_pred = KNN.predict(X\_test) |
| --- |

**code 24**

| # Imprimir los resultados de la validación cruzada  print("Accuracy: %0.2f (+/- %0.2f)" % (scores\_train.mean(), scores\_train.std() \* 2))  print(classification\_report(y\_test, y\_test\_pred))  # Classification report  from sklearn.metrics import classification\_report  import seaborn as sns  import matplotlib.pyplot as plt  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()  visual\_classification\_report(y\_test, y\_test\_pred) |
| --- |

**code 25**

| #Plot confusion matrix  cm = confusion\_matrix(y\_test, y\_test\_pred)  classes = np.unique(y\_test)  plt.figure(figsize=(8, 6))  sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=classes, yticklabels=classes)  plt.xlabel('y\_pred')  plt.ylabel('y\_test')  plt.title('Matriz de Confusión')  plt.show() |
| --- |

**code 26**

| # Accuracy de los datos tanto del train como del test.  print("Accuracy - Datos de train: {} " .format(accuracy\_score(y\_train, y\_train\_pred)))  print("Accuracy - Datos de test: {} ". format(accuracy\_score(y\_test, y\_test\_pred)))  # Barrido de k vecinos.  train\_score = []  test\_score = []  k\_value = list(range(2, 30))  for i in k\_value:  KNN = KNeighborsClassifier(n\_neighbors=i)  KNN.fit(X\_train,y\_train)  y\_train\_pred = KNN.predict(X\_train)  train\_score.append(accuracy\_score(y\_train, y\_train\_pred))  y\_test\_pred = KNN.predict(X\_test)  test\_score.append(accuracy\_score(y\_test, y\_test\_pred))  plt.figure(figsize=(10,6))  plt.plot(k\_value, train\_score, color="blue", label="accuracy train")  plt.plot(k\_value, test\_score, color="red", label="accuracy test")  plt.legend()  plt.xlabel('K Value')  plt.ylabel('Accuracy')  plt.xticks(range(min(k\_value), max(k\_value)+1, 1))  plt.show() |
| --- |

**code 27**

| import os  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from IPython.display import set\_matplotlib\_formats  set\_matplotlib\_formats('retina')  import seaborn as sns  import tensorflow as tf  from tensorflow import keras  from sklearn.model\_selection import train\_test\_split  directorio\_principal = r'/content/drive/MyDrive/BigData/'  espectrogramas = []  directorios = [] # Lista para almacenar los nombres de los directorios  for carpeta in os.listdir(directorio\_principal):  carpeta\_path = os.path.join(directorio\_principal, carpeta)  # Si la ruta es un directorio  if os.path.isdir(carpeta\_path):  # Iterar sobre los archivos en la carpeta actual  for archivo in os.listdir(carpeta\_path):  archivo\_path = os.path.join(carpeta\_path, archivo)  # Cargar el espectrograma desde el archivo .npy sin aplanar  espectrograma = np.load(archivo\_path)  espectrogramas.append(espectrograma) # Agregar el espectrograma sin aplanar  directorios.append(carpeta) # Agregar el nombre del directorio  # Convertir la lista de espectrogramas en un DataFrame  df\_espectrogramas = pd.DataFrame({'Espectrograma': espectrogramas, 'Directorio': directorios})  # Save df\_espectrogramas  main\_folder = "/content/drive/MyDrive/BigData/"  df\_espectrogramas.to\_parquet(os.path.join(main\_folder, "df\_espectrogramas.parquet"), index=False)  # Read df\_espectrogramas  main\_folder = "/content/drive/MyDrive/BigData/"  df\_espectrogramas = pd.read\_parquet(os.path.join(main\_folder, "df\_espectrogramas.parquet"))  df\_espectrogramas.tail(2) |
| --- |

**code 28**

| # Directorio principal donde están los archivos .npy  directorio\_principal = r'/content/drive/MyDrive/BigData/'  main\_folder = "/content/drive/MyDrive/BigData/"  # Parámetros para el procesamiento por lotes  batch\_size = 250 # Número de espectrogramas a procesar por lote  temp\_files = [] # Lista para almacenar los nombres de archivos temporales  # Función para procesar y guardar un lote de datos  def process\_and\_save\_batch(espectrogramas, directorios, batch\_index):  espectrograma\_df = pd.DataFrame(espectrogramas)  espectrograma\_df.columns = [f'Espectrograma\_{i}' for i in range(espectrograma\_df.shape[1])]  df\_batch = pd.concat([espectrograma\_df, pd.DataFrame({'Directorio': directorios})], axis=1)    # Guardar el lote en un archivo Parquet temporal  temp\_file = os.path.join(main\_folder, f"temp\_batch\_{batch\_index}.parquet")  df\_batch.to\_parquet(temp\_file, index=False)  temp\_files.append(temp\_file)  # Inicializar variables  espectrogramas = []  directorios = []  batch\_index = 0  # Recorrer las carpetas en el directorio principal  for carpeta in os.listdir(directorio\_principal):  carpeta\_path = os.path.join(directorio\_principal, carpeta)  # Si la ruta es un directorio  if os.path.isdir(carpeta\_path):  # Iterar sobre los archivos en la carpeta actual  for archivo in os.listdir(carpeta\_path):  archivo\_path = os.path.join(carpeta\_path, archivo)  # Cargar el espectrograma desde el archivo .npy  espectrograma = np.load(archivo\_path)  # Aplanar el espectrograma y agregarlo a la lista  espectrogramas.append(espectrograma.flatten())  # Agregar el nombre del directorio a la lista  directorios.append(carpeta)    # Si alcanzamos el tamaño del lote, procesar y guardar  if len(espectrogramas) == batch\_size:  process\_and\_save\_batch(espectrogramas, directorios, batch\_index)  espectrogramas = []  directorios = []  batch\_index += 1  # Procesar y guardar cualquier dato restante  if espectrogramas:  process\_and\_save\_batch(espectrogramas, directorios, batch\_index)  # Combinar todos los archivos Parquet temporales en uno solo  final\_table = pa.concat\_tables([pq.read\_table(temp\_file) for temp\_file in temp\_files])  pq.write\_table(final\_table, os.path.join(main\_folder, "df\_espectrogramas\_final.parquet"))  # Eliminar archivos temporales  for temp\_file in temp\_files:  os.remove(temp\_file)  print("Proceso completado y archivo Parquet final guardado.") |
| --- |

**code 29**

| # Contar la cantidad de datos de cada "Directorio"  count\_per\_species = df\_espectrogramas["Directorio"].value\_counts().sort\_index()  # Crear el diagrama de barras de cantidad de datos por "Directorio"  plt.figure(figsize=(12, 6))  plt.barh(count\_per\_species.index, count\_per\_species.values)  plt.ylabel("Directorio")  plt.xlabel("Cantidad de datos")  plt.title("Cantidad de datos por Directorio")  plt.tight\_layout()  plt.show() |
| --- |

**code 30**

| # Seleccionamos los datos de los primeras 9 especies  Primeros5directorios = ["Ardea purpurea","Gallinula chloropus","Acrocephalus scirpaceus","Anas platyrhynchos","Circus aeruginosus"]  df = df\_espectrogramas[df\_espectrogramas['Directorio'].isin(Primeros5directorios)]  # Separamos los datos de los espectrogramas según los datos X e y. El target es adivinar el nombre de la especie, o sea el directorio.  X = df.drop(columns=['Directorio'])  y = df["Directorio"]  # Dividir en conjuntos de entrenamiento y prueba  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=1)  # Verificar la forma de los datos de X\_train y X\_test  print(f"X\_train shape: {X\_train.shape}")  print(f"X\_test shape: {X\_test.shape}")    # Diccionario de mapeo  species\_dict = {  "Acrocephalus arundinaceus": 1,  "Acrocephalus melanopogon": 2,  "Acrocephalus scirpaceus": 3,  "Alcedo atthis": 4,  "Anas (Mareca) strepera": 5,  "Anas platyrhynchos": 6,  "Ardea purpurea": 7,  "Botaurus stellaris": 8,  "Charadrius alexandrinus": 9,  "Ciconia ciconia": 10,  "Circus aeruginosus": 11,  "Coracias garrulus": 12,  "Dendrocopos minor": 13,  "Fulica atra": 14,  "Gallinula chloropus": 15,  "Himantopus himantopus": 16,  "Ixobrychus minutus": 17,  "Motacilla flava": 18,  "Porphyrio porphyrio": 19,  "Tachybaptus ruficollis": 20}  # Cambiamos los nombres en y por números  y\_train\_num = np.array([species\_dict[species] for species in y\_train])  y\_test\_num = np.array([species\_dict[species] for species in y\_test])  # Asegurarse de que las imágenes tienen 4 dimensiones  train\_images = X\_train[..., np.newaxis] / 255.0 if X\_train.ndim == 3 else X\_train / 255.0  test\_images = X\_test[..., np.newaxis] / 255.0 if X\_test.ndim == 3 else X\_test / 255.0  # Verificar la forma de los datos  print(f"train\_images shape: {train\_images.shape}")  print(f"test\_images shape: {test\_images.shape}")  import tensorflow as tf  from tensorflow.keras import layers, models  # Escalamos las imágenes entre 0 y 1  train\_images = X\_train / 255  test\_images = X\_test / 255  # Definir la arquitectura del modelo con 3 capas ocultas de 5 neuronas  model = models.Sequential([  layers.Dense(5, activation='relu', input\_shape=(train\_images.shape[1],)),  layers.Dense(5, activation='relu'),  layers.Dense(5, activation='relu'),  layers.Dense(20, activation='softmax') # Capa de salida con 20 neuronas (20 clases)  ])  # Compilar el modelo  model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Entrenar el modelo  history = model.fit(train\_images, y\_train\_num, validation\_split=0.1, batch\_size=256, epochs=10)  # Batch\_size son cuantas imágenes pasan en grupo por la red de neuronas + backtrain hasta pasar todas las imágenes que se tienen  # El resultado de dividir Nºtotal imagen/batch\_size son cuantas pasadas ha tenido que hacer y es lo que nos indica en la salida abajo de Epoch que pone NUM/NUM    # Filtrar y convertir a etiquetas numéricas usando species\_dict  y\_test\_num\_filtered = np.array([species\_dict[species] for species in y\_test if species in Primeros5directorios])  test\_label\_predictions\_filtered = np.array([species\_dict[species] for species in y\_test if species in Primeros5directorios])  # Matriz de confusión para conjunto de prueba  cm = confusion\_matrix(y\_test\_num\_filtered, test\_label\_predictions\_filtered)  # Obtener las etiquetas para mostrar en la matriz de confusión  labels = [species\_dict[species] for species in Primeros5directorios]  # Visualizar la matriz de confusión  disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)  plt.figure(figsize=(10, 8))  disp.plot(cmap=plt.cm.Blues)  plt.title('Matriz de Confusión')  plt.show()  # Accuracy  accuracy = model.evaluate(train\_images, y\_train\_num)  print(accuracy[1], "Accuracy en Train")  accuracy = model.evaluate(test\_images, y\_test\_num)  print(accuracy[1], "Accuracy en Test")  # Evolución de la accuracy para cada época  plt.plot(history.history['accuracy'])  plt.plot(history.history['val\_accuracy'])  plt.title('Model accuracy')  plt.ylabel('accuracy')  plt.xlabel('epoch')  plt.legend(['train', 'val'], loc='upper left')  plt.show()  # Evolución de la loss para cada época  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('Model loss')  plt.ylabel('loss')  plt.xlabel('epoch')  plt.legend(['train', 'val'], loc='upper left')  plt.show() |
| --- |

**code 31**

| n = 250 # Parámetro número de datos máximo por directorio  # Calcular la cantidad de datos por Directorio  count\_per\_species = df\_espectrogramas['Directorio'].value\_counts()  # Identificar los directorios que tienen más y menos de n datos  directorios\_con\_mas\_de\_n = count\_per\_species[count\_per\_species > n].index  directorios\_con\_menos\_de\_n = count\_per\_species[count\_per\_species < n].index  # Crear una máscara booleana para seleccionar aleatoriamente n filas de esos directorios  random\_samples = []  for dir in directorios\_con\_mas\_de\_n:  indices = df\_espectrogramas[df\_espectrogramas['Directorio'] == dir].index # Listado de índices de datos con ese nombre de dir  random\_samples.extend(np.random.choice(indices, size=min(n, len(indices)), replace=False)) # Números al azar  # Crear df con los espectogramas de mas de n cortados y concatenar los menores de n  df = df\_espectrogramas[df\_espectrogramas.index.isin(random\_samples)]  df = pd.concat([df, df\_espectrogramas[df\_espectrogramas['Directorio'].isin(directorios\_con\_menos\_de\_n)]], ignore\_index=True)  # Graficar cantidad de datos de cada "Directorio"  count\_per\_species = df["Directorio"].value\_counts().sort\_index()  plt.figure(figsize=(12, 6))  plt.barh(count\_per\_species.index, count\_per\_species.values)  plt.ylabel("Directorio")  plt.xlabel("Cantidad de datos")  plt.title("Cantidad de datos por Directorio")  plt.tight\_layout()  plt.show()  df.tail(2)  # Separamos los datos de los espectrogramas según los datos X e y. El target es adivinar el nombre de la especie, o sea el directorio.  X = df.drop(columns=['Directorio'])  y = df["Directorio"]  # Dividir en conjuntos de entrenamiento y prueba  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=1)  # Verificar la forma de los datos de X\_train y X\_test  print(f"X\_train shape: {X\_train.shape}")  print(f"X\_test shape: {X\_test.shape}")  # Cambiamos los nombres en y por números  y\_train\_num = np.array([species\_dict[species] for species in y\_train])  y\_test\_num = np.array([species\_dict[species] for species in y\_test])  import tensorflow as tf  from tensorflow.keras import layers, models  # Escalamos las imágenes entre 0 y 1  train\_images = X\_train / 255  test\_images = X\_test / 255  # Definir la arquitectura del modelo con 3 capas ocultas de 5 neuronas  model = models.Sequential([  layers.Dense(5, activation='relu', input\_shape=(train\_images.shape[1],)),  layers.Dense(5, activation='relu'),  layers.Dense(5, activation='relu'),  layers.Dense(20, activation='softmax') # Capa de salida con 20 neuronas (20 clases)  ])  # Compilar el modelo  model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Entrenar el modelo  history = model.fit(train\_images, y\_train\_num, validation\_split=0.1, batch\_size=256, epochs=10)  # Batch\_size son cuantas imágenes pasan en grupo por la red de neuronas + backtrain hasta pasar todas las imágenes que se tienen  # El resultado de dividir Nºtotal imagen/batch\_size son cuantas pasadas ha tenido que hacer y es lo que nos indica en la salida abajo de Epoch que pone NUM/NUM    from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  # Filtrar y convertir a etiquetas numéricas usando species\_dict  all\_species = [  "Acrocephalus arundinaceus",  "Acrocephalus melanopogon",  "Acrocephalus scirpaceus",  "Alcedo atthis",  "Anas (Mareca) strepera",  "Anas platyrhynchos",  "Ardea purpurea",  "Botaurus stellaris",  "Charadrius alexandrinus",  "Ciconia ciconia",  "Circus aeruginosus",  "Coracias garrulus",  "Dendrocopos minor",  "Fulica atra",  "Gallinula chloropus",  "Himantopus himantopus",  "Ixobrychus minutus",  "Motacilla flava",  "Porphyrio porphyrio",  "Tachybaptus ruficollis"]  y\_test\_num\_filtered = np.array([species\_dict[species] for species in y\_test if species in all\_species])  test\_label\_predictions\_filtered = np.array([species\_dict[species] for species in y\_test if species in all\_species])  # Matriz de confusión para conjunto de prueba  cm = confusion\_matrix(y\_test\_num, test\_label\_predictions\_filtered)  # Obtener las etiquetas para mostrar en la matriz de confusión  labels = [species\_dict[species] for species in all\_species]  # Visualizar la matriz de confusión  disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)  plt.figure(figsize=(10, 8))  disp.plot(cmap=plt.cm.Blues)  plt.title('Matriz de Confusión')  plt.show()  # Accuracy  accuracy = model.evaluate(train\_images, y\_train\_num)  print(accuracy[1], "Accuracy en Train")  accuracy = model.evaluate(test\_images, y\_test\_num)  print(accuracy[1], "Accuracy en Test")  # Evolución de la accuracy para cada época  plt.plot(history.history['accuracy'])  plt.plot(history.history['val\_accuracy'])  plt.title('Model accuracy')  plt.ylabel('accuracy')  plt.xlabel('epoch')  plt.legend(['train', 'val'], loc='upper left')  plt.show()  # Evolución de la loss para cada época  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('Model loss')  plt.ylabel('loss')  plt.xlabel('epoch')  plt.legend(['train', 'val'], loc='upper left')  plt.show() |
| --- |

**code 32**

| p = 70 # Porcentaje de información a eliminar  # Selección columnas que tienen el prefijo "Espectrograma"  espectograma\_columns = [col for col in df\_espectrogramas.columns if col.startswith("Espectrograma")]  # Determinar cuántas columnas corresponden al p%  num\_to\_remove = int(len(espectograma\_columns) \* p/100)  # Seleccionar aleatoriamente las columnas que se eliminarán, equiespaiadas  indices = np.linspace(0, len(espectograma\_columns) - 1, num\_to\_remove, dtype=int)  columns\_to\_remove = [espectograma\_columns[i] for i in indices]  # Eliminar las columnas seleccionadas del DataFrame  df\_espectrogramas\_reduced = df\_espectrogramas.drop(columns=columns\_to\_remove)  # Verificar la eliminación  print("Número de columnas originales:", len(df\_espectrogramas.columns))  print("Número de columnas después de eliminar el 70%:", len(df\_espectrogramas\_reduced.columns))  Después el código es igual que **code 31** con df=df\_espectrogramas\_reduced y restando 1 en y\_train\_num en estas líneas:  history = model.fit(train\_images, y\_train\_num-1, validation\_split=0.1, batch\_size=256, epochs=10)  accuracy = model.evaluate(train\_images, y\_train\_num-1)  accuracy = model.evaluate(test\_images, y\_test\_num-1) |
| --- |

**code 33**

| # Realizamos la clasificación para distintos parámetros para encontrar los óptimos  MLPC = MLPClassifier(early\_stopping = True,max\_iter=1000,activation='relu', batch\_size = 200, solver='adam',random\_state=1)  parameters = {  'hidden\_layer\_sizes': [(5,5,5),(20,20),(100,100,100)],  'learning\_rate\_init' : [0.01, 0.00001]  }  from sklearn.model\_selection import GridSearchCV  grid\_search = GridSearchCV(MLPC, parameters, cv=5, verbose = True)  grid\_search.fit(X\_train, y\_train)  means = grid\_search.cv\_results\_['mean\_test\_score']  stds = grid\_search.cv\_results\_['std\_test\_score']  for mean, std, params in zip(means, stds, grid\_search.cv\_results\_['params']):  print("He obtenido una accuracy de %0.3f (+/-%0.03f) con los parámetros %r" % (mean, std \* 2, params)) |
| --- |

**code 34**

| from sklearn.metrics import accuracy\_score  # Entrenamiento con todo el dataset de train sacando precisión del test.  new\_MPLC = MLPClassifier(\*\*grid\_search.best\_params\_,max\_iter=1000,activation='relu',  batch\_size = 200, solver='adam',random\_state=1)  new\_MPLC.fit(X\_train, y\_train)  predicciones = new\_MPLC.predict(X\_test)  print("Accuracy en test: ", accuracy\_score(predicciones, y\_test) )  print("Iteraciones realizadas por el modelo =", len(new\_MPLC.loss\_curve\_))  plt.plot(new\_MPLC.loss\_curve\_, color='Blue')  plt.xlabel('Iterations')  plt.ylabel('Loss')  plt.title('Loss Curve')  plt.show() |
| --- |

**code 35**

| # Parece que la pérdida seguiría decreciendo con más iteraciones.  # Vamos a realizar algunas iteraciones más.  # Se añaden los parámetros: tol = 0, n\_iter\_no\_change=1000  new2\_MPLC = MLPClassifier(\*\*grid\_search.best\_params\_, early\_stopping = True,  max\_iter=1000,activation='relu', batch\_size = 200,  solver='adam',random\_state=1, tol = 0, n\_iter\_no\_change=1000)  new2\_MPLC.fit(X\_train, y\_train)  predicciones2 = new2\_MPLC.predict(X\_test)  print("Accuracy en test: ", accuracy\_score(predicciones2, y\_test) )  print("Iteraciones realizadas por el modelo 2 =", len(new2\_MPLC.loss\_curve\_))  plt.plot(new2\_MPLC.loss\_curve\_, color='Red')  plt.xlabel('Iterations')  plt.ylabel('Loss')  plt.title('Loss Curve')  plt.show() |
| --- |

**code 36**

| print("Iteraciones realizadas por el modelo 2 =", len(new2\_MPLC.loss\_curve\_))  # Graficar solo hasta las primeras 100 iteraciones  plt.plot(new2\_MPLC.loss\_curve\_[:100], color='Red')  plt.xlabel('Iterations')  plt.ylabel('Loss')  plt.title('Loss Curve (First 100 Iterations)')  plt.show() |
| --- |

**code 37**

| import os  import librosa  import numpy as np  import pandas as pd  # Parámetros  audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  max\_length = 1 # Segundos que van a tener todos los audios  FRAME\_SIZE = 2048  HOP\_LENGTH = 512  def root\_mean\_squared(y, frame\_size, hop\_length):  rms = librosa.feature.rms(y=y, frame\_length=frame\_size, hop\_length=hop\_length)  return np.mean(rms, axis=1)  def amplitude\_mean\_max\_values(y, frame\_size, hop\_length):  amplitude = np.abs(librosa.stft(y, n\_fft=frame\_size, hop\_length=hop\_length))  mean\_amplitude = np.mean(amplitude, axis=1)  max\_amplitude = np.max(amplitude, axis=1)  return mean\_amplitude, max\_amplitude  def calcular\_caracteristicas(y, sr, FRAME\_SIZE, HOP\_LENGTH, audio\_file, bird\_name):  # Calcular las características del audio  mfcc = librosa.feature.mfcc(y=y, n\_mfcc=13, sr=sr)  mfcc\_scaled = np.mean(mfcc, axis=1)  delta1\_mfcc = librosa.feature.delta(mfcc\_scaled, mode='nearest')  delta2\_mfcc = librosa.feature.delta(mfcc\_scaled, mode='nearest', order=2)  zcr = librosa.feature.zero\_crossing\_rate(y=y)  rms = root\_mean\_squared(y, FRAME\_SIZE, HOP\_LENGTH)  amplitude\_mean, amplitude\_max = amplitude\_mean\_max\_values(y, FRAME\_SIZE, HOP\_LENGTH)  spectral\_centroid = librosa.feature.spectral\_centroid(y=y, sr=sr, n\_fft=FRAME\_SIZE, hop\_length=HOP\_LENGTH)  spectral\_flux = librosa.onset.onset\_strength(S=librosa.feature.melspectrogram(y=y, sr=sr, n\_fft=FRAME\_SIZE, hop\_length=HOP\_LENGTH))  spectral\_bandwidth = librosa.feature.spectral\_bandwidth(y=y, sr=sr, n\_fft=FRAME\_SIZE, hop\_length=HOP\_LENGTH)  features = {  'audio\_file': audio\_file,  'bird\_name': bird\_name,  'zcr': np.mean(zcr),  'rms': np.mean(rms),  'spectral\_centroid': np.mean(spectral\_centroid),  'spectral\_flux': np.mean(spectral\_flux),  'spectral\_bandwidth': np.mean(spectral\_bandwidth),  'mfcc': np.mean(mfcc\_scaled),  'delta1\_mfcc': np.mean(delta1\_mfcc),  'delta2\_mfcc': np.mean(delta2\_mfcc),  'amplitude\_mean': np.mean(amplitude\_mean),  'amplitude\_max': np.mean(amplitude\_max)  }  return features  def extract\_audio\_features(audio\_folder):  audio\_features\_data = []  # Iterar sobre las carpetas de las especies  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  bird\_name = species\_folder # Asumiendo que el nombre de la especie es el nombre de la carpeta  # Iterar sobre los archivos de audio dentro de cada carpeta de especies  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE \* 2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  if len(y) < max\_length \* sr: # Si el audio es más pequeño que 1 segundo, repetir el audio  repeats = (max\_length \* sr) // len(y) + 1 # Número de repeticiones necesarias  y = np.tile(y, repeats)[:max\_length \* sr] # Repetir y recortar  # Calcular las características del audio  audio\_feature\_dict = calcular\_caracteristicas(y, sr, FRAME\_SIZE, HOP\_LENGTH, audio\_file, bird\_name)  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  continue  audio\_features\_data.append(audio\_feature\_dict)  df = pd.DataFrame(audio\_features\_data)  return df  df = extract\_audio\_features(audio\_folder)  df.tail(2)  # Save df\_delta  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_repetir.parquet"), index=False) |
| --- |

**code 38**

| ## CODE 10  # Leer delta, sin mfcc  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df\_delta = pd.read\_parquet(os.path.join(main\_folder, "df\_repetir.parquet"))  df\_delta.tail(2)  ## CODE 11  audio\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/Audio\_Clip/"  data = []  FRAME\_SIZE = 1024  # Dataframe con el nombre de la especie (directorio) y del archivo de audio  for species\_folder in os.listdir(audio\_folder):  species\_folder\_path = os.path.join(audio\_folder, species\_folder)  if os.path.isdir(species\_folder\_path):  for audio\_file in os.listdir(species\_folder\_path):  audio\_path = os.path.join(species\_folder\_path, audio\_file)  try:  y, sr = librosa.load(audio\_path)  if len(y) < FRAME\_SIZE\*2: # Si el audio es más pequeño que 2 ventanas de análisis, no se analiza.  continue  data.append((species\_folder, audio\_file))  except Exception as e:  print(f"Error al cargar el archivo {audio\_file}: {e}")  df\_names = pd.DataFrame(data, columns=['Directorio', 'NombreArchivo'])  df\_names.tail(2)  ## CODE 12  # Eliminar la terminación ".wav"  df\_names['NombreArchivo'] = df\_names['NombreArchivo'].str.replace('.wav', '')  df = pd.concat([df\_names, df\_delta], axis=1)  df.tail(2)  # Save Data Frame usado en el Machine Learning  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df.to\_parquet(os.path.join(main\_folder, "df\_repetir\_ml.parquet"), index=False)  ## CODE 15  # Leer datos  main\_folder = "/content/drive/MyDrive/BigData/DataMining/Practica1/"  df = pd.read\_parquet(os.path.join(main\_folder, "df\_repetir\_ml.parquet"))  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import classification\_report  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score  from sklearn.metrics import confusion\_matrix  from sklearn.ensemble import RandomForestClassifier  import warnings  warnings.filterwarnings(action="default")  df.tail(2) |
| --- |

**code 39**

| from sklearn.model\_selection import train\_test\_split  X = df.drop(columns=['Directorio', 'bird\_name', 'audio\_file', 'NombreArchivo', 'AudioArchivo'])  y = df['Directorio']  # Separar los nombres únicos en conjuntos de entrenamiento y prueba  unique\_audio\_files = df['AudioArchivo'].unique()  train\_audio\_files, test\_audio\_files = train\_test\_split(unique\_audio\_files, test\_size=0.35, random\_state=42)  # Crear máscaras para filtrar el DataFrame original  train\_mask = df['AudioArchivo'].isin(train\_audio\_files)  test\_mask = df['AudioArchivo'].isin(test\_audio\_files)  # Crear conjuntos de entrenamiento y prueba  X\_train = X[train\_mask]  y\_train = y[train\_mask]  X\_test = X[test\_mask]  y\_test = y[test\_mask]  # Verificar la separación  train\_size = X\_train.shape[0]  test\_size = X\_test.shape[0]  total\_size = df.shape[0]  train\_percentage = (train\_size / total\_size) \* 100  test\_percentage = (test\_size / total\_size) \* 100  print(f'Tamaño del conjunto de entrenamiento: {train\_size} ({train\_percentage:.2f}%)')  print(f'Tamaño del conjunto de prueba: {test\_size} ({test\_percentage:.2f}%)')    # Diccionario de mapeo  species\_dict = {  "Acrocephalus arundinaceus": 1,  "Acrocephalus melanopogon": 2,  "Acrocephalus scirpaceus": 3,  "Alcedo atthis": 4,  "Anas strepera": 5,  "Anas platyrhynchos": 6,  "Ardea purpurea": 7,  "Botaurus stellaris": 8,  "Charadrius alexandrinus": 9,  "Ciconia ciconia": 10,  "Circus aeruginosus": 11,  "Coracias garrulus": 12,  "Dendrocopos minor": 13,  "Fulica atra": 14,  "Gallinula chloropus": 15,  "Himantopus himantopus": 16,  "Ixobrychus minutus": 17,  "Motacilla flava": 18,  "Porphyrio porphyrio": 19,  "Tachybaptus ruficollis": 20}  # Cambiamos los nombres en y por números  y\_train\_num = np.array([species\_dict[species] for species in y\_train])  y\_test\_num = np.array([species\_dict[species] for species in y\_test]) |
| --- |

**code 40**

| ## CODE 17  # Entrenar el modelo de Random Forest con diferentes números de árboles y calcular el error de predicción  n\_trees = 100  errors = []  for i in range(1, n\_trees + 1):  rf = RandomForestClassifier(n\_estimators=i, random\_state=42)  rf.fit(X\_train, y\_train\_num)  error = 1 - rf.score(X\_test, y\_test\_num)  errors.append(error)  # Graficar el error de predicción en función del número de árboles  plt.figure(figsize=(10, 6))  plt.plot(range(1, n\_trees + 1), errors, marker='o', linestyle='-')  plt.xlabel('Número de árboles')  plt.ylabel('Error de predicción')  plt.title('Error de predicción del Random Forest')  plt.grid(True)  plt.show()  RF = RandomForestClassifier(max\_depth=20, n\_estimators = 200, random\_state=0)  RF.fit(X\_train, y\_train\_num)  y\_train\_pred = RF.predict(X\_train)  y\_test\_pred = RF.predict(X\_test)  ## CODE 18  # Gráfica de las features  number\_of\_features = list(range(X\_train.shape[1]))  print("Feature importances Random forest:", RF.feature\_importances\_)  plt.barh(number\_of\_features, RF.feature\_importances\_)  plt.yticks(number\_of\_features, X.columns)  plt.xlabel("Feature Importances")  plt.ylabel("Features")  plt.show()  # Accuracy  print("Accuracy - Datos de train: {} " .format(accuracy\_score(y\_train, y\_train\_pred)))  print("Accuracy - Datos de test: {} ". format(accuracy\_score(y\_test, y\_test\_pred)))  from sklearn.preprocessing import LabelEncoder  # Instanciar el LabelEncoder  label\_encoder = LabelEncoder()  # Convertir las etiquetas de clase (y\_train y y\_test) a valores numéricos  y\_train\_encoded = label\_encoder.fit\_transform(y\_train)  y\_test\_encoded = label\_encoder.transform(y\_test)  # Asegurarse de que y\_train\_pred y y\_test\_pred sean de tipo numérico adecuado (int)  y\_train\_pred = y\_train\_pred.astype(int)  y\_test\_pred = y\_test\_pred.astype(int)  # Verificar tipos de datos después de la conversión  print("Tipo de y\_train\_encoded:", y\_train\_encoded.dtype)  print("Tipo de y\_test\_encoded:", y\_test\_encoded.dtype)  print("Tipo de y\_train\_pred:", y\_train\_pred.dtype)  print("Tipo de y\_test\_pred:", y\_test\_pred.dtype)  # Calcular accuracy y visualizar el classification\_report  print("Accuracy - Datos de train: {:.2f}".format(accuracy\_score(y\_train\_encoded, y\_train\_pred)))  print("Accuracy - Datos de test: {:.2f}".format(accuracy\_score(y\_test\_encoded, y\_test\_pred)))  from sklearn.metrics import classification\_report  import seaborn as sns  import matplotlib.pyplot as plt  # 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()  visual\_classification\_report(y\_test\_num, y\_test\_pred)  ## CODE 19  # Matriz de confusión  conf\_matrix = confusion\_matrix(y\_test\_num, y\_test\_pred)  classes = np.unique(y\_test)  plt.figure(figsize=(8, 6))  sns.heatmap(conf\_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=classes, yticklabels=classes)  plt.xlabel('y\_pred')  plt.ylabel('y\_test')  plt.title('Matriz de Confusión')  plt.show() |
| --- |