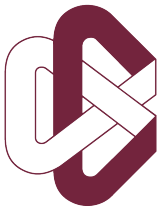


CIMAT

Centro de Investigación en Matemáticas
Unidad Monterrey

Análisis Multimodal

Tarea 2

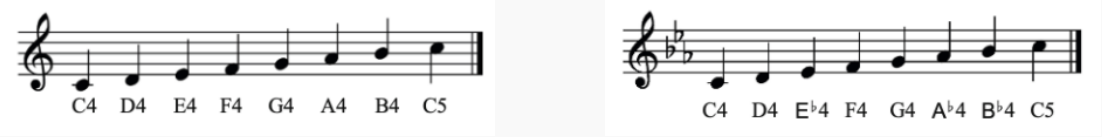
Gustavo Hernández Angeles

28 de octubre de 2025

Índice

Ejercicio 1:

En la clase vimos cómo obtener las frecuencias centrales (en Hz) de diferentes notas o tonalidades. Calcula éstas frecuencias para todas las notas de la escala Do mayor (Figura 1) y Do menor (Figura 1).



(a) (b)

Figura 1.1: (a) Escala C-Mayor. (b) Escala C-Menor. Ambas iniciando en C4.

1.1 Inciso a): Escala C-Mayor

El patrón de intervalos a partir de la tónica es: Tono, Tono, Semitono, Tono, Tono, Tono, Semitono. Recordando que C4 tiene un número MIDI de 60, podemos obtener los números MIDI de las notas en la escala C-Mayor haciendo saltos según el patrón de intervalos (2 para tonos y 1 para semitonos), por lo que la secuencia resultante en números MIDI sería 60, 62, 64, 65, 67, 69, 71, 72.

Luego, utilizando la fórmula para calcular la frecuencia de una nota a partir de su número MIDI (Eq. 1.1), podemos obtener las frecuencias de las notas en la escala C-Mayor (Tabla 1.1). Aquí notamos que la nota A4 (N. MIDI 69) tiene una frecuencia de 440 Hz, que es el estándar de afinación.

$$F(p) = 440 \times 2^{\frac{(p-69)}{12}} \tag{1.1}$$

Nota	**N. MIDI (p)**	**Freq. (Hz)**
C4	60	≈ 261.63
D4	62	≈ 293.66
E4	64	≈ 329.63
F4	65	≈ 349.23
G4	67	≈ 392.00
A4	69	440.00
B4	71	≈ 493.88
C5	72	≈ 523.25

Cuadro 1.1: Frecuencias de la Escala Do Mayor (C-Mayor) usando números MIDI.

1.2 Inciso b): Escala C-Menor

En la escala C-Menor, el patrón de intervalos a partir de la tónica es: Tono, Semitono, Tono, Tono, Semitono, Tono, Tono, obteniendo la secuencia 60, 62, 63, 65, 67, 68, 70, 72, en números MIDI. Siguiendo el mismo procedimiento que en el inciso anterior, podemos obtener los números MIDI y las frecuencias correspondientes para las notas en la escala C-Menor (Tabla 1.2). Notamos que la nota Ab4 (N. MIDI 68) tiene una frecuencia de aproximadamente 415.30 Hz, que es un semitono por debajo de A4 (440 Hz), por lo que es consistente con la estructura de la escala menor.

Nota	**N. MIDI (p)**	**Freq. (Hz)**
C4	60	≈ 261.63
D4	62	≈ 293.66
Eb4	63	≈ 311.13
F4	65	≈ 349.23
G4	67	≈ 392.00
Ab4	68	≈ 415.30
Bb4	70	≈ 466.16
C5	72	≈ 523.25

Cuadro 1.2: Frecuencias de la Escala Do Menor (C-Menor) usando números MIDI.

Ejercicio 2:

La Figura 2.1 muestra la forma de onda de un audio de los primeros 8 segundos de la quinta sinfonía de Beethoven (puedes en el extracto en formato mp3 en el moodle).

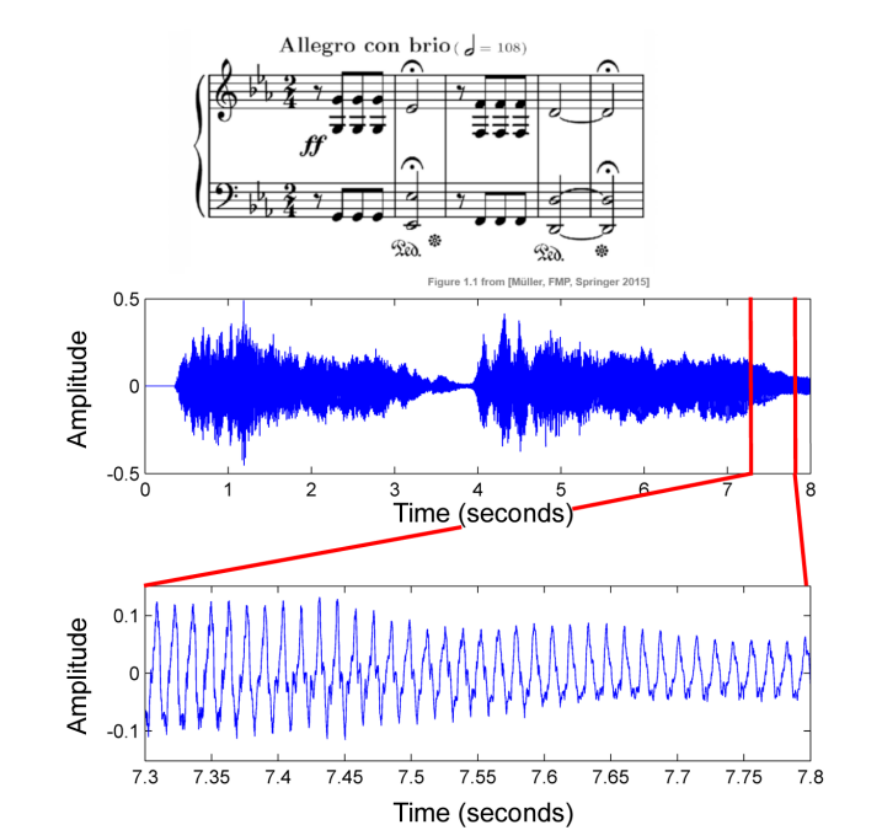


Figura 2.1: Partitura con los primeros cinco compases de la Sinfonía No. 5 de Beethoven, y su correspondiente señal, como forma de onda, que abarca los primeros 8 segundos.

a) Estima la frecuencia fundamental del sonido registrado en la sección que abarca del segundo 7.3 al 7.8, contando el número de ciclos de oscilación.

b) Determina a qué nota musical corresponde como lo vimos en clase, es decir, buscando aquella nota cuya frecuencia fundamental es más cercana a la que estimaste en el inciso anterior. ¿Tiene sentido según la partitura mostrada en la figura 2.1?

2.1 Inciso a)

Al inspeccionar la forma de onda ampliada para el intervalo $t \in [7.3, 7.8]$ segundos, se puede realizar un conteo visual de los ciclos de oscilación. En esta ventana de tiempo, cuya duración es $\Delta t = 7.8\text{ s} - 7.3\text{ s} = 0.5\text{ s}$, se observan aproximadamente $N = 36.5$ ciclos.

La frecuencia fundamental f_0 se estima como la cantidad de ciclos por unidad de tiempo:

$$f_0 = \frac{N \text{ ciclos}}{\Delta t \text{ (s)}} = \frac{36.5 \text{ ciclos}}{0.5 \text{ s}} = 73 \text{ Hz} \tag{2.1}$$

La frecuencia fundamental estimada es, por lo tanto, $f_0 \approx 73$ Hz.

2.2 Inciso b)

Para identificar la nota musical correspondiente a $f_0 \approx 73$ Hz, la comparamos con las frecuencias estándar del temperamento igual (con afinación $A_4 = 440$ Hz). La frecuencia teórica de la nota D_2 (Re 2) es $f_{D_2} \approx 73.42$ Hz, siendo esta la nota cuya frecuencia fundamental es más cercana a nuestra estimación.

El segmento de audio analizado ($t \in [7.3, 7.8]$ s) corresponde al decaimiento del último acorde mostrado, el cual se ubica en el quinto compás y está marcado con un calderón.

- En el pentagrama superior (Clave de Sol): La nota es D_4 (Re 4).
- En el pentagrama inferior (Clave de Fa): Las notas son D_2 (Re 2) y D_3 (Re 3).

La frecuencia fundamental f_0 de un sonido complejo generalmente corresponde a la frecuencia de la nota más grave que lo compone. Las otras notas presentes en el acorde, D_3 y D_4, son octavas de D_2. Sus frecuencias teóricas ($f_{D_3} \approx 146.8$ Hz y $f_{D_4} \approx 293.6$ Hz) son múltiplos enteros (armónicos) de la frecuencia de D_2 ($f_{D_3} \approx 2 \times f_{D_2}$; $f_{D_4} \approx 4 \times f_{D_2}$).

Por lo tanto, **si tiene sentido** que la frecuencia fundamental estimada ($f_0 \approx 73$ Hz) corresponda a la nota D_2 (Re 2), ya que esta es la nota más grave del acorde tocado en el compás 5, que es el segmento de audio analizado.

Ejercicio 3:

Definimos $f : \mathbb{R} \to \Gamma$ como la función de cuantización de una señal continua a un conjunto de valores discretos $\Gamma \in \mathbb{R}$. La función f más simple es la cuantización uniforme, donde asigna un valor de amplitud $a \in \mathbb{R}$, un valor cuantizado mediante:

$$f(a) = \text{sign}(a)\Delta \left\lfloor \frac{|a|}{\Delta} + \frac{1}{2} \right\rfloor \tag{3.1}$$

donde Δ es el tamaño del paso de cuantización, y $\lfloor \cdot \rfloor$ es la función piso. La diferencia entre la señal original y la señal cuantizada se conoce como **error de cuantización**. Muchas veces es más conveniente definir los niveles de cuantización λ en un rango limitado de amplitud $[-a_{\min}, a_{\max}]$, en lugar del tamaño de paso Δ. En este caso, $\Delta = \frac{|a_{\max} - a_{\min}|}{\lambda - 1}$.

a) Escribe una función que implementa la cuantización uniforme (Eq. 3.1).

b) Obtén la gráfica de la cuantización para $a = g(x; \theta) \in \mathbb{R}$, con g la función lineal y una sinusoidal con los parámetros vistos en clase. Gráfica también el error absoluto de cuantización correspondiente.

c) Con el proceso de cuantización, la señal se codifica en alguno de los $\lambda = 2^b$ valores de amplitud igualmente espaciados en el intervalo definido, donde cada intervalo es de tamaño Δ. El parámetro b es el número de bits necesario para codificar la señal, y puede obtenerse mediante $b = \log_2(\lambda)$. Aunque es posible definir cualquier número de niveles de cuantización para la codificación, es muy común definirlos mediante el número de bits, por ejemplo 8 bits (256 niveles), 16 bits (65,536 niveles), etc. Para tener una referencia, el sonido en un CD está codificado generalmente con una cuantización de 16 bits. Realiza la cuantización uniforme de alguna(s) señal(es) de audio que se encuentran en la plataforma usando 8, 6, 4 y 2 bits. ¿Qué puedes notar en la señal con las diferentes cuantizaciones?

3.1 Inciso a)

En Python, la función para implementar la cuantización uniforme según la ecuación 3.1 puede ser escrita de la siguiente manera:

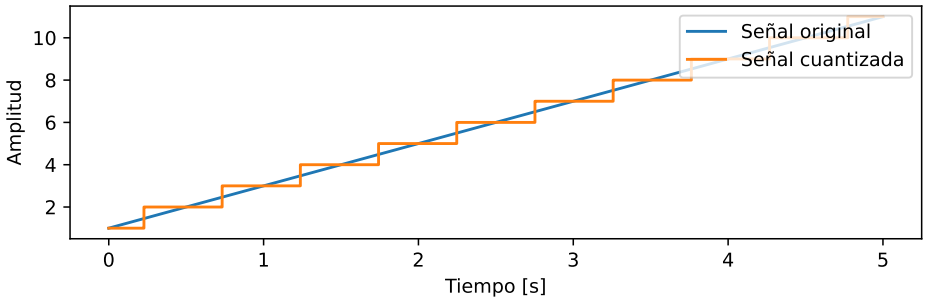
```
def f(a:float,
  delta:float) -> float:
return np.sign(a) * delta * np.floor(np.abs(a)/delta + 0.5)
```

En donde se hace uso de la librería NumPy para las operaciones matemáticas.

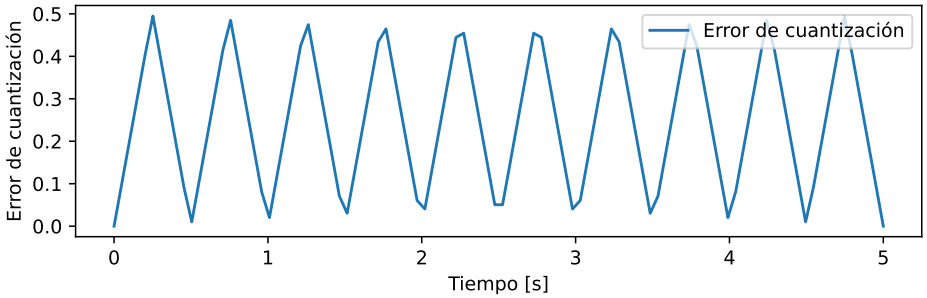
3.2 Inciso b)

Función lineal

Para la función lineal, podemos definir $g(x; \theta) = mx + c$, donde m es la pendiente y c es la intersección con el eje y. Por ejemplo, podemos tomar $m = 2$ y $c = 1$.



(a)



(b)

Figura 3.1: (a) Cuantización de una función lineal y (b) su error absoluto de cuantización.

En la figura 3.1 se muestra la señal original y su versión cuantizada (con $\Delta = 1$), así como una gráfica para el error absoluto de cuantización.

Podemos observar que la señal cuantizada aproxima la señal original, pero con saltos discretos, cuya resolución depende del valor de Δ. El error absoluto de cuantización muestra cómo la diferencia oscila entre 0 y $\Delta/2 = 0.5$ de forma lineal, lo cual es esperado.

Función sinusoidal

En la figura 3.2 se muestra la señal original y su versión cuantizada (con $\Delta = 0.5$), así como una gráfica para el error absoluto de cuantización. En este caso utilizamos la función sinusoidal $g(x;\theta) = \sin(x)$. Aquí también podemos observar que la señal cuantizada aproxima la señal original. El error absoluto de cuantización muestra un patrón periódico, oscilando entre 0 y $\Delta/2 = 0.25$, lo cual es consistente con la naturaleza periódica de la señal original. Observamos un patrón de error similar al lineal para las regiones entre picos y valles, mientras que en los picos y valles muestra una gráfica en forma de "U".

3.3 Inciso c)

En este problema se implementaron dos funciones principales: una para obtener el valor de Δ (`obtener_delta`) basado en el número de bits y la señal original haciendo uso de la fórmula propuesta

$$\Delta = \frac{a_{max} - a_{min}}{\lambda - 1}$$

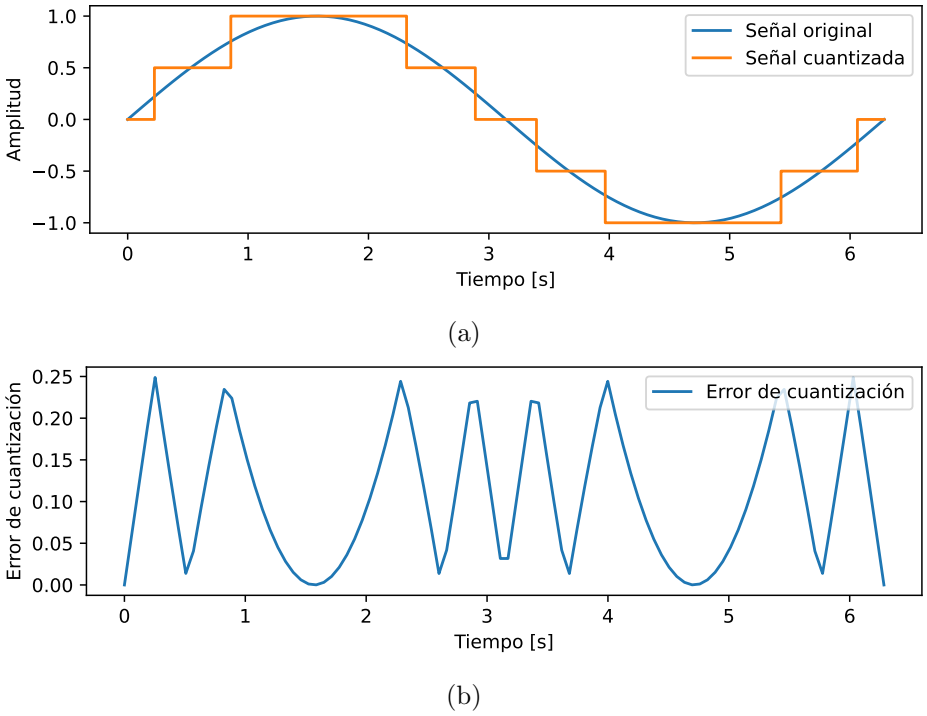


Figura 3.2: (a) Cuantización de una función sinusoidal y (b) su error absoluto de cuantización.

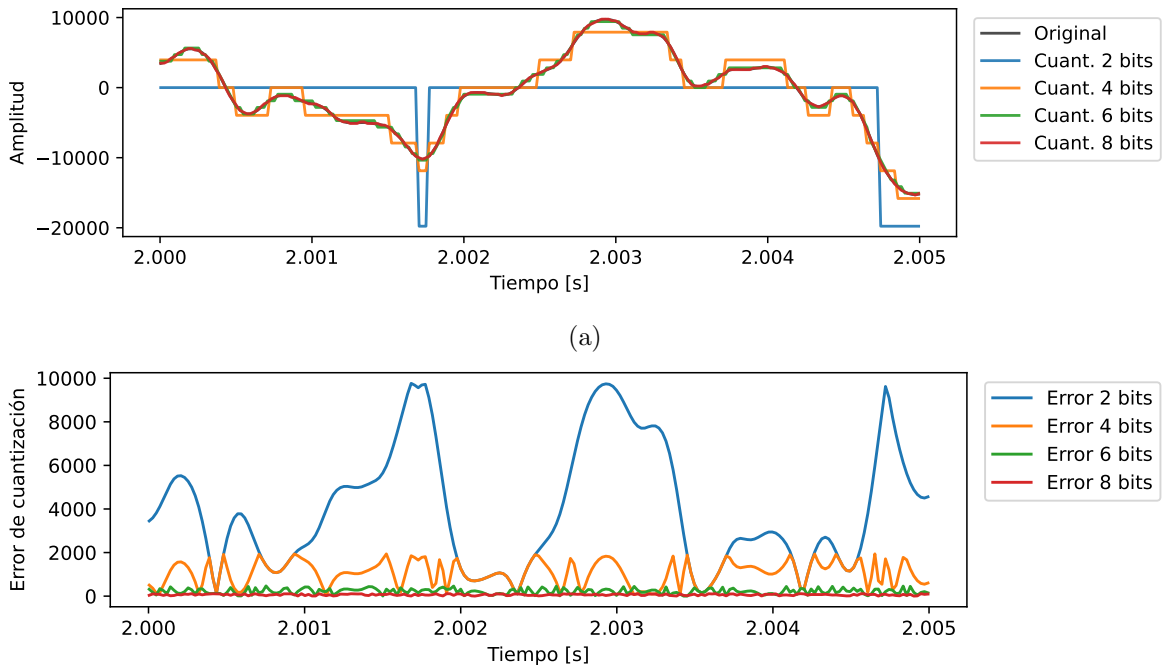
, y la segunda (`obtener_cuantizada`) para realizar la cuantización uniforme de una señal de audio dada la ruta del archivo y el número de bits b.

Nos enfocaremos en el archivo de audio `major.wav` para ilustrar los resultados obtenidos con diferentes niveles de cuantización (8, 6, 4 y 2 bits). Este archivo de audio contiene una señal musical con dos canales, lo cual es importante a considerar al momento de la cuantización. En nuestro caso, se trabajó eligiendo el Δ basado en ambos canales, es decir, se consideró un valor máximo a_{max} de la señal y mínimo global a_{min} para ambos canales al calcular Δ, con la finalidad de mantener las amplitudes relativas entre los canales. Para las gráficas, se seleccionó un segmento de la señal para una mejor visualización (2s a 2.005s) y el canal izquierdo.

En la figura 3.3 se muestran las gráficas de la señal original y las señales cuantizadas para cada nivel de bits, junto con sus respectivos errores absolutos de cuantización. Observamos que los errores de cuantización disminuyen conforme aumenta el número de bits, lo cual es consistente con la teoría de cuantización; a medida que se incrementa el número de bits, la resolución de la cuantización mejora (dado por $\lambda = 2^b$), resultando en una aproximación más precisa de la señal original. La gráfica de las señales (Figura 3.3a) muestra cómo la señal cuantizada se acerca más a la señal original a medida que aumentamos el número de bits.

El caso extremo de 2 bits muestra una señal muy distorsionada, mostrando solo capacidad para representar 3 niveles de intensidad en el segmento analizado, lo cual resulta en una pérdida significativa de información y calidad sonora. Mientras que la cuantización con 8 bits ofrece una representación mucho más fiel de la señal original, con un error de cuantización considerablemente menor, casi nulo.

En cuanto a calidad de sonido, destaco que no cuento con equipo de audio decente para el



(a)

(b)

Figura 3.3: (a) Cuantización de la señal musical *major.wav* y (b) su error absoluto de cuantización. Ambas gráficas muestran las señales cuantizadas para 8, 6, 4 y 2 bits para el canal izquierdo.

análisis y sin embargo, las diferencias son muy notorias. La cuantización con 8 bits suena bastante similar a la señal original, resultándome imperceptible la diferencia. A partir de la cuantización con 6 bits, se empieza a notar severamente la pérdida de calidad, con un sonido más áspero y con la introducción de ruido. Las cuantizaciones con 4 y 2 bits resultan en una calidad de sonido muy pobre, con distorsiones graves y pérdida de detalles musicales, haciendo que la música sea prácticamente irreconocible en el caso de 2 bits (además de que suena muy fuerte, quizá dañe bocinas).

Ejercicio 4:

Los datos de la carpeta `midi_selected` contiene archivos `midi` de diferentes obras de 8 compositores. Aunque la música se compone de diversos elementos, en éste ejercicio consideraremos sólo 3: notas, acordes y silencios. La función `get_all_notes` del script `fun.py` extrae los acordes, notas y silencios (con su respectiva duración), de los archivos `midi`.

a) Crea un corpus de las obras musicales de todos los compositores. Considera como variable dependiente y al compositor, y crea conjuntos de entrenamiento y prueba estratificado por ésta variable de respuesta. ¿Cómo es la distrbución de las obras por compositor? Usa algunos gráficos informativos de las características del corpus.

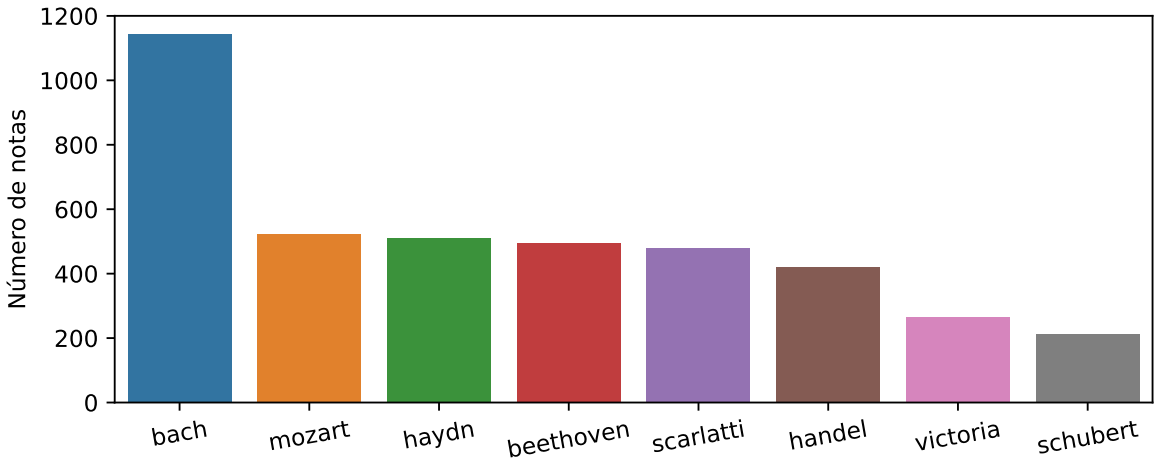
b) Usando una representación TF-IDF del corpus, implementa al menos 3 clasificadores para estimar y. Utiliza algún método eficiente para el ajuste de los modelos, asegurando una buena generalización. Elabora un resúmen breve de tus resultados incluyendo todas las métricas de desempeño, los criterios usados para la representación del corpus, el ajuste de los modelos, tus hallazgos, y demás información que creas pertinente. ¿Qué mejoras sugieres para tener un mejor resultado?

4.1 Inciso a)

Crear el corpus fue sencillo gracias a la función `get_all_notes` proporcionada, la cual extrae las notas, acordes y silencios de los archivos MIDI con espacios de separación. Se iteró sobre todas las carpetas del archivo `midi_selected` aplicando dicha función, y se almacenaron los resultados en un directorio llamado `notes`, donde cada archivo de texto corresponde a una obra musical de un compositor espeífico, dado por su prefijo. Finalmente, se creó un archivo CSV que contiene dos columnas: `notes` y `author`. La columna `notes` contiene la secuencia de notas, acordes y silencios extraídos, mientras que la columna `author` indica el compositor correspondiente.

Los conjuntos de **entrenamiento** y **prueba** se crearon utilizando la función `train_test_split` de la librería `sklearn.model_selection`, con la opción de estratificación basada en la variable dependiente `author`. Se asignó un 80 % de los datos al conjunto de entrenamiento y un 20 % al conjunto de prueba, asegurando que la distribución de obras por compositor se mantuviera en ambos conjuntos.Lo anterior resultó en un conjunto de entrenamiento que contiene un total de 4050 obras.

La distribución de obras por compositor en el corpus (Figura 4.1) muestra que el compositor *Bach* tiene una cantidad significativamente mayor de obras en comparación con otros; apróximadamente 1100 obras. Esta distribución desigual puede influir en el rendimiento de los clasificadores, ya que los modelos podrían estar sesgados hacia los compositores con más datos disponibles. Los compositores *Victoria* y *Schubert* tienen la menor cantidad de obras, con alrededor de 250 cada uno. Esta disparidad puede afectar la capacidad del modelo para aprender patrones representativos de estos compositores. Los demás compositores tienen cantidades intermedias de obras (450), y algo similares entre sí, lo que podría facilitar un aprendizaje más equilibrado para esos casos.



(a)

Figura 4.1: Distribución de obras por compositor en el conjunto de entrenamiento.

4.2 Inciso b)

4.2.1 Representación numérica de las obras.

El conjunto de datos creado posee una estructura de texto, donde cada obra musical está representada como una secuencia de notas, acordes y silencios, acompañados de su duración. Para convertir este corpus en una representación númerica adecuada para el entrenamiento de clasificadores, se utilizó la representación TF-IDF, cuya implementación se realizó mediante la clase `TfidfVectorizer` de la librería `scikit-learn`. Los argumentos principales utilizados fueron:

- `ngram_range=(1,3)`: Considera n-gramas de longitud 1 a 3, capturando así secuencias de notas y acordes.
- `token_pattern="\b [\w .##-]+\b "`: Quizá lo más importante, este patrón de tokenización permite capturar notas musicales con alteraciones (sostenidos y bemoles), así como acordes y silencios, con sus respectivas duraciones.

Se experimentó con parámetros cuyo proposito es reducir la dimensionalidad del espacio de características, como `max_df` y `min_df`, pero no se observaron mejoras significativas en el rendimiento de los clasificadores (de hecho, empeoraban). Por lo tanto, se optó por no utilizarlos en la versión final del modelo.

4.2.2 Random Forest

Hiperparámetros

El primer clasificador implementado fue un **Random Forest**, utilizando la implementación de `scikit-learn`. El procedimiento para obtener el mejor modelo involucró el uso de `Optuna` para la optimización de hiperparámetros, librería que permite una búsqueda eficiente en el espacio de hiperparámetros mediante técnicas de optimización bayesiana [1]. Además, se empleó un seguimiento de los experimentos utilizando `MLflow` [2], lo que facilitó la gestión y comparación

de los diferentes modelos entrenados. El flujo de trabajo puede enlistarse de la siguiente manera:

1. Definición del espacio de búsqueda de hiperparámetros, incluyendo parámetros como el número de árboles (`n_estimators`), la profundidad máxima de los árboles (`max_depth`), el número mínimo de muestras por hoja (`min_samples_leaf`) y el número mínimo de muestras para dividir un nodo (`min_samples_split`).
2. Configuración de un estudio de `Optuna` para optimizar los hiperparámetros basándose en la métrica de F1-score macro obtenida mediante validación cruzada k-fold (con $k = 5$).
3. Con MLflow, se recogen los 3 mejores modelos de la búsqueda según la métrica de F1-score macro, se entrenan en el conjunto completo de entrenamiento, y se evalúan en el conjunto de prueba.
4. Se selecciona el modelo con el mejor desempeño en el conjunto de prueba para el análisis final.

Cuadro 4.1: Hiperparámetros optimizados para Random Forest con 20 ensayos.

Hiperparámetro	**Rango de búsqueda**	**Valor óptimo**
`n_estimators`	[100, 1000] (paso 100)	400
`max_depth`	[10, 50] (paso 5)	30
`min_samples_split`	[2, 10]	5
`min_samples_leaf`	[1, 5]	2

En la Tabla 4.1 se presentan los hiperparámetros optimizados para el modelo de Random Forest, junto con sus respectivos rangos de búsqueda y los valores óptimos encontrados. Automatizar este proceso fue importante para asegurar que el modelo estuviera bien ajustado a los datos, maximizando su capacidad de generalización.

Resultados

En la Tabla 4.2 se resume el desempeño del Random Forest en el conjunto de prueba. Globalmente, el modelo alcanza una exactitud de 0.74, con F1 macro de 0.72 y F1 ponderado de 0.73, lo que indica un rendimiento equilibrado ante el desbalance moderado del corpus.

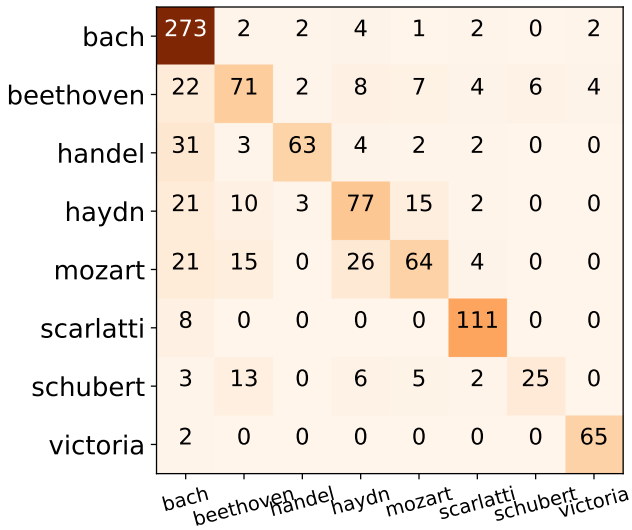


Figura 4.2: Matriz de confusión del clasificador Random Forest en el conjunto de prueba.

Cuadro 4.2: Resultados por clase y métricas globales del Random Forest en prueba.

Clase	**Precisión**	**Recall**	**F1**	**Soporte**
bach	0.72	0.95	0.82	286
beethoven	0.62	0.57	0.60	124
handel	0.90	0.60	0.72	105
haydn	0.62	0.60	0.61	128
mozart	0.68	0.49	0.57	130
scarlatti	0.87	0.93	0.90	119
schubert	0.81	0.46	0.59	54
victoria	0.92	0.97	0.94	67
Accuracy		0.74		1013
Macro	0.77	0.70	0.72	1013
Weighted	0.74	0.74	0.73	1013

De acuerdo con la Tabla 4.2, los autores que el modelo distingue mejor son *victoria* y *scarlatti*, lo que sugiere que sus patrones son más distintivos para el modelo. *bach* presenta un recall muy alto (0.95) pero una precisión moderada (0.72), consistente con posibles confusiones hacia esta clase debidas a su mayor soporte. En contraste, *mozart* (F1=0.49) y *schubert* (F1=0.46) muestran los menores recalls, reflejando mayor confusión entre autores del periodo clásico.

Estos resultados apuntan a dos frentes de mejora: representación y balance de clases. Una ampliación de la representación (p. ej., n-gramas más largos o rasgos rítmicos/duración) podría ayudar a separar mejor a los compositores clásicos; y técnicas de balanceo (re-muestreo o `class_weight`) aliviarían el sesgo hacia clases con mayor soporte.

4.2.3 XGBoost

Hiperparámetros

El segundo clasificador implementado fue **XGBoost** (Extreme Gradient Boosting) [3], conocido por su eficiencia y rendimiento en tareas de clasificación. Al igual que con Random Forest, se utilizó `Optuna` para la optimización de hiperparámetros y `MLflow` para el seguimiento de experimentos. El flujo de trabajo fue similar al descrito anteriormente, con la definición del espacio de búsqueda de hiperparámetros específicos para XGBoost, como la tasa de aprendizaje (`learning_rate`), el parámetro de regularización (`Gamma`) y la profundidad máxima de los árboles (`max_depth`). Para la estimación del error durante la optimización, se utilizó también la métrica de F1-score macro sobre el conjunto de prueba.

Cuadro 4.3: Hiperparámetros optimizados para XGBoost con 20 ensayos.

Hiperparámetro	**Rango de búsqueda**	**Valor óptimo**
`learning_rate`	[0.01, 0.4] (paso 0.01)	0.01
`max_depth`	[1, 10]	7
`Gamma`	[0, 5] (paso 0.5)	1

En la Tabla 4.3 se presentan los hiperparámetros optimizados para el modelo de XGBoost, junto con sus respectivos rangos de búsqueda y los valores óptimos encontrados. La optimización cuidadosa de estos hiperparámetros es crucial para maximizar el rendimiento del modelo en la tarea de clasificación de compositores. En esta ocasión, la optimización también ayudó a suavizar las exigencias computacionales del modelo, permitiendo un entrenamiento más eficiente.

Resultados

En la Tabla 4.4 se resume el desempeño de XGBoost en el conjunto de prueba. Globalmente, el modelo alcanza una exactitud de 0.79, con F1 macro de 0.78 y F1 ponderado de 0.79, superando al Random Forest (0.74 de exactitud y 0.72 de F1 macro).

Cuadro 4.4: Resultados por clase y métricas globales de XGBoost en prueba.

Clase	**Precisión**	**Recall**	**F1**	**Soporte**
bach	0.89	0.90	0.89	286
beethoven	0.70	0.63	0.66	124
handel	0.78	0.85	0.81	105
haydn	0.65	0.75	0.70	128
mozart	0.60	0.58	0.59	130
scarlatti	0.96	0.92	0.94	119
schubert	0.74	0.63	0.68	54
victoria	1.00	0.97	0.98	67
Accuracy		0.79		1013
Macro	0.79	0.78	0.78	1013
Weighted	0.79	0.79	0.79	1013

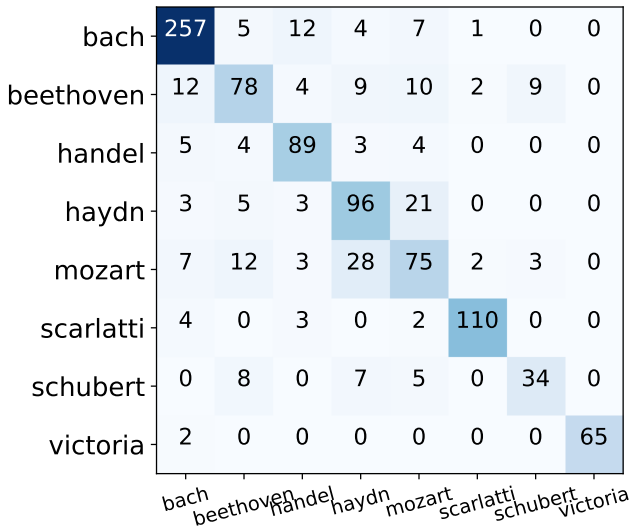


Figura 4.3: Matriz de confusión del clasificador XGBoost en el conjunto de prueba.

De acuerdo con la Tabla 4.4, las clases mejor modeladas son *victoria* (F1=0.98) y *scarlatti* (F1=0.94), seguidas de *bach* (F1=0.89). También se observan mejoras respecto a Random Forest en autores clásicos como *haydn* (F1=0.70) y *schubert* (F1=0.68). Las clases con menor desempeño siguen siendo *mozart* (F1=0.59) y *beethoven* (F1=0.66), lo que sugiere patrones más sutiles o mayor confusión entre compositores del periodo clásico.

4.2.4 Kernel SVM

Hiperparámetros

En esta ocasión, se utilizó un clasificador basado en **Support Vector Machines** (SVM) con kernel radial (RBF). Se empleó la implementación de `scikit-learn` para SVM. En esta ocasión, decidí irme por los parámetros predeterminados del modelo (i.e. `C=1.0`, `gamma='scale'`), ya que la optimización de hiperparámetros para SVM resultó ser computacionalmente costosa (el entrenamiento tiene una duración de 15 minutos). A pesar de ello, el modelo mostró un rendimiento razonable en las pruebas iniciales, por lo que se optó por no realizar una búsqueda exhaustiva de hiperparámetros en esta ocasión.

Resultados

En la Tabla 4.5 se resume el desempeño del SVM con kernel RBF en prueba. El modelo alcanza exactitud 0.80, F1 macro 0.79 y F1 ponderado 0.80, ligeramente por encima de XGBoost.

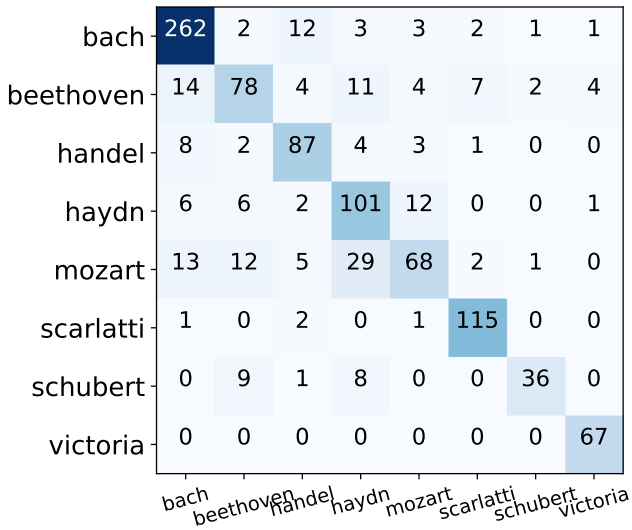


Figura 4.4: Matriz de confusión del clasificador SVM (RBF) en el conjunto de prueba.

Cuadro 4.5: Resultados por clase y métricas globales de SVM (RBF) en prueba.

Clase	**Precisión**	**Recall**	**F1**	**Soporte**
bach	0.86	0.92	0.89	286
beethoven	0.72	0.63	0.67	124
handel	0.77	0.83	0.80	105
haydn	0.65	0.79	0.71	128
mozart	0.75	0.52	0.62	130
scarlatti	0.91	0.97	0.93	119
schubert	0.90	0.67	0.77	54
victoria	0.92	1.00	0.96	67
Accuracy		0.80		1013
Macro	0.81	0.79	0.79	1013
Weighted	0.80	0.80	0.80	1013

De acuerdo con la Tabla 4.5, *victoria* (F1=0.96) y *scarlatti* (F1=0.93) destacan, seguidas de *bach* (F1=0.89). Persisten dificultades relativas en *mozart* (F1=0.62) y *beethoven* (F1=0.67), con mejoras notables en *haydn* (recall 0.79) y *schubert* (F1=0.77).

4.2.5 Conclusión

En términos globales, el SVM con kernel RBF obtiene el mejor desempeño (Accuracy = 0.80, F1 macro = 0.79), seguido muy de cerca por XGBoost (0.79/0.78) y, finalmente, Random Forest (0.74/0.72). La superioridad de SVM es pequeña pero consistente en las métricas agregadas.

Autores con mejores y peores clasificaciones:

- Mejores: *victoria* y *scarlatti* son sistemáticamente las clases más sólidas en los tres

modelos (XGBoost y SVM alcanzan F1 entre 0.93–0.98). *bach* también presenta F1 alto, especialmente con SVM (0.89).

- Peores: En todos los modelos, las clases más retadoras son *mozart* y *beethoven*. Con XGBoost y SVM sus F1 rondan 0.59–0.67; en Random Forest el rendimiento es menor en general. *schubert* mejora notablemente con SVM (F1=0.77) respecto a RF (0.59). *haydn* incrementa su *recall* con SVM (0.79) frente a RF y también mejora con XGBoost (F1=0.70).

Eficiencia computacional y requerimientos:

- Tiempo de entrenamiento: el modelo más tardado fue Kernel SVM con aproximadamente **15 minutos**. Tanto XGB como RF entrenaron en el orden de **1 minuto**.
- Exigencia de hardware: XGB aprovechó GPU, lo que acelera el entrenamiento pero aumenta el uso de memoria de video durante la construcción de árboles; en equipos con GPU modesta puede requerir ajustar profundidad de árbol y bins. En CPU sigue siendo competitivo y estable para este corpus.

En conjunto, Kernel SVM entrega el mejor resultado absoluto a mayor costo computacional; XGBoost ofrece el mejor balance entre precisión y tiempo (y escala bien con GPU); Random Forest queda detrás en métricas pero es el más simple y robusto en recursos.

Ejercicio 5:

En la tarea anterior, leíste un ensayo de Daphne Oram. Entonces eres de los pocos afortunados que la conoce, ya que su legado como inventora, compositora y pionera de música electrónica, es (tristemente) desconocido por la mayoría.

Explora un poco más sobre ella y su legado. Puedes ver el contenido de `daphneoram.org`. Enfócate principalmente en *oramics*, su invento, y describe si lo que has aprendido hasta ahora en el curso, te ayuda a entender y valorar su ingenio. Realiza un breve reporte sobre estos tópicos.

Más allá de su rol como cofundadora del BBC Radiophonic Workshop, el legado más profundo y visionario de Daphne Oram es, sin duda, su invención: *Oramics*. Este sistema no fue solo una herramienta musical; fue la manifestación física de los principios fundamentales del procesamiento de señales, décadas antes de que las interfaces gráficas de usuario y las Estaciones de Trabajo de Audio Digital (DAWs) se convirtieran en estándar.

5.1 Oramics

El concepto central de *Oramics* es la "música dibujada". En esencia, Oram diseñó una máquina que traducía información gráfica directamente en sonido. El sistema utilizaba una serie de tiras de película de 35mm sobre las cuales Oram dibujaba o "pintaba". Estas películas pasaban frente a células fotoeléctricas que leían los patrones dibujados.

La genialidad radica en cómo abstrajo los componentes del sonido en elementos visuales:

- **Forma de Onda (Timbre):** El aspecto más directo. Oram dibujaba la forma de la onda (el "grano" del sonido) en una sección de la película. Una línea suave y ondulada (cercana a una sinusoide) produciría un tono puro, mientras que una forma de onda compleja, como un diente de sierra o una cuadrada, generaría un timbre rico en armónicos.
- **Tono (Pitch):** Controlado gráficamente, a menudo mediante máscaras o "plantillas"que barrían el espectro de frecuencias.
- **Amplitud (Volumen):** La intensidad o grosor de las líneas dibujadas controlaba la amplitud de la señal generada.
- **Envolvente (ADSR):** Oram también diseñaba "máscaras"que modulaban la amplitud a lo largo del tiempo, creando efectivamente envolventes de ataque, decaimiento, sostenimiento y relajación de forma puramente gráfica.

5.2 Entendiendo Oramics a través del Curso

Sin una comprensión del procesamiento de señales, *Oramics* podría parecer casi mágico o puramente artístico. Sin embargo, lo que hemos aprendido nos permite entenderlo como un sistema de ingeniería y síntesis de audio increíblemente avanzado.

El curso nos ha enseñado que cualquier señal periódica, sin importar cuán compleja sea su forma de onda (su timbre), puede descomponerse en una suma de ondas sinusoidales simples. Oram invirtió este concepto: en lugar de descomponer, ella realizaba una **síntesis gráfica**. Al dibujar una forma de onda compleja, ella estaba definiendo implícitamente la amplitud y

fase de todos los armónicos que compondrían ese sonido. Estaba, literalmente, dibujando el espectro de Fourier en el dominio del tiempo.

El ingenio de Daphne Oram no fue solo "dibujar sonidos bonitos". Su genialidad fue diseñar y construir un sistema físico que implementaba los principios más profundos de la teoría de señales. Ella entendió que la frecuencia, la amplitud y la forma de onda (timbre) no eran conceptos musicales abstractos, sino parámetros físicos y matemáticos de una señal que podían ser controlados y manipulados.

Oramics demuestra que un osciloscopio y un sintetizador pueden ser, en esencia, la misma herramienta. El curso nos proporciona el lenguaje formal para apreciar que Daphne Oram no solo fue una compositora, sino también una ingeniera de señales que trabajaba con luz y película en lugar de código.

Referencias

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] Matei A. Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41:39–45, 2018.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.