Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo Fin de Grado
Grado en Ingeniería Electrónica

Estudio, desarrollo y evaluación de técnicas de aprendizaje automático para la identificación de notas, instrumentos y/o compositores en archivos de música

Clasificación de compositores de música clásica

Autor:
Lucas Pardo Bernardi
Director:
Luis Javier Rodriguez Fuentes

Leioa, 23 de Junio de 2022

Índice general

Capítulo 1

Introducción

En los últimos años los algoritmos de aprendizaje automático (en inglés, *machine learning*) han ganado mucha popularidad debido al enorme desarrollo y avances en el tema. Esto ha llevado a que se utilicen en distintos campos y para una gran variedad de tareas. Estos algorítmos son tan flexibles y eficientes que se acaban utilizando en casi cualquier tarea de regresión o clasificación. Uno de los campos con mayor popularidad es el de reconocimiento y clasificación audiovisual.

El uso de algoritmos de aprendizaje automático para el reconocimiento de audio es algo que se lleva aplicando desde hace varias décadas para el reconocimiento y clasificación tanto de voces humanas como de música. La forma de obtener parámetros cuantificables de una señal acústica es bastante sencilla, y disponemos de la herramienta necesaria desde hace 200 años, la transformada de Fourier. Sin embargo, relacionar de forma eficaz estos parámetros con otros más complejos o no cuantificables ha sido difícil hasta la aparición del aprendizaje automático.

En la actualidad, el término machine learning (abreviadamente, ML) se utiliza ampliamente en el mundo de la música para clasificación de géneros y autores, detección de instrumentos, sistemas de recomendación e incluso para la creación de música.

1.1. Objetivo

Este trabajo se centrará en la tarea de clasificación de compositores. Para ello se utilizarán dos conjuntos de datos (datasets): MAESTRO [1] y MusicNet [2]. Ambos conjuntos están compuestos por canciones de música clásica de dominio público. MAESTRO contiene 1276 canciones interpretadas con un único modelo específico de piano mientras que MusicNet contiene 330 canciones interpretadas con distintas cantidades y configuraciones de instrumentos (cuartetos de piano, quintetos de violín, etc.).

Se empezará explorando la forma de obtener distintos parámetros de una señal acústica y cuáles pueden ser útiles para la tarea propuesta. Para calcular los parámetros se utilizará la librería de Python llamada *librosa* [3] que ya tiene implementada las funciones necesarias para calcular parámetros de un archivo de audio (en nuestro caso, archivos *.mkv*).

Finalmente, se explorarán distintos algoritmos de ML y su eficacia al clasificar los archivos de ambos conjuntos. Para ello se utilizará la librería de Python llamada *Scikit-*

learn [4] que contiene una colección bastante extensa de algoritmos de ML y funciones complementarias para la transformación de datos.

1.2. Metodología

El trabajo se desarollará en su totalidad utilizando el lenguaje de programación *Python*. No se hará mención directa al código, sólo a algunas funciones utilizadas y su propósito. Los códigos utilizados se encuentran disponibles en este repositorio [1]. Dentro de las carpetas correspondientes ("Maestro", "musicnet") se encuentran los ficheros *.csv* de los metadatos y los rasgos extraídos y utilizados en este trabajo. Los distintos códigos se pueden ejecutar directamente siempre que las carpetas "Maestro" y "musicnet"se encuentren en el mismo directorio, en caso contrario hay que cambiar las referencias al directorio dentro del código (Figura 1.1). Por otra parte, cada código dispone al principio de una serie de parámetros como el conjunto o el modelo (Figura 1.1). Sin embargo, para cambiar los hiperparámetros de cada modelo como *k*, *weights*, *C*, etc., es necesario hacerlo manualmente en el código correspondiente. En el repositorio se encuentra un archivo de texto llamado "Descripción" con una breve descripción de cada código disponible.

```
import numpy as np
import pandas as pd
from sklearn import svm, metrics, neighbors, multioutput, multiclass, linear_mod
                    discriminant_analysis, kernel_ridge, gaussian_process, cross
                    model_selection, calibration, neural_network
#from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Parámetros

```
dataset = "musicnet"          ### Conjunto a utilizar
duration = 60                 ### Duración del troceado
showDistribution = False      ### Mostrar distribución Train-Test
usedModel = "knn"             ### modelo a utilizar: {knn, svc, gpc, vc}

if dataset.lower() == "musicnet":
    #### MusicNet:
    features = pd.read_csv("musicnet/features" + str(duration) + ".csv")
else:
    #### Maestro:
    features = pd.read_csv("Maestro/features" + str(duration) + ".csv")
```

Directorio

Figura 1.1: Principio del fichero *main.py*.

[1] https://drive.google.com/drive/folders/1AO9Ka-LR2EWH5aDuDwnw8EmyHlELRwYP?usp=sharing

Capítulo 2

Procesamiento de Música

En *machine learning* prácticamente cualquier medida obtenida a partir de un objeto se puede utilizar como medio de representación de dicho objeto en un proceso de clasificación. Esto significa que incluso medidas como la duración de una canción, por muy irrelevante que nos parezca, puede ser útil para encontrar sesgos en ciertas clasificaciones. Por ejemplo, la duración media de las canciones de un cierto compositor A podría ser de tres minutos mientras que las de un compositor B podrían rondar los cinco minutos, lo cual nos indicaría que si nos encontramos con una nueva canción de aproximadamente tres minutos de duración, hay muchas probabilidades de que dicha canción haya sido compuesta por el compositor A. En la realidad, las cosas no son tan fáciles y sesgos tan evidentes como el del ejemplo anterior no son comunes. Por ello, tan sólo usar la duración de una canción como rasgo representativo es en realidad completamente inviable, dado que no sólo hay innumerables canciones de distintos compositores con duraciones similares sino que incluso un mismo compositor puede tener obras de duraciones tan distintas como tres y doce minutos. Por lo tanto, se hace necesario encontrar otras medidas o características que puedan ser útiles en la tarea de clasificación musical. En este capítulo trataremos de explicar cómo obtener nuevas características representativas y por qué pueden resultar útiles para la tarea de clasificación.

2.1. Características de una Canción

Desde el punto de vista físico, un sonido no es más que una onda acústica, una vibración que se propaga por el aire. Esta vibración tiene una frecuencia o, de forma más general, un espectro de frecuencias. Este espectro de frecuencias es lo que determina completamente la forma en la que percibimos este sonido. Un adulto promedio puede percibir frecuencias entre veinte y veinte mil Hz pero debido al funcionamiento logarítmico de la mente humana, dos frecuencias cuyo ratio sea una potencia de dos (2, 4, 8, ...) se perciben de forma extremadamente similar. Esto es lo que se conoce como una *octava*. Desde el siglo XVIII hasta la actualidad, la música occidental (de origen europeo) adoptó un sistema estándar de doce clases tonales con el que componer canciones llamado *temperamento igual de 12 tonos* (TET por sus siglas en inglés). Esto implica que de todo el rango audible de un adulto, sólo doce frecuencias geométricamente espaciadas se consideran como distintas clases tonales, de forma que sus múltiplos potencia de dos forman todo el conjunto de tonos utilizados, llamados *notas*. Utilizando esta definición se pueden calcular las frecuencias de todos los tonos a partir de una única frecuencia a la que llamamos *tono estándar* que usualmente se toma como $f_0 = 440$ Hz. A esta clase tonal se le llama **la** o **A** (notación americana). Como hemos explicado anteriormente, $2f_0 = 880$ Hz se corresponde

a una octava y por lo tanto también pertenece a la misma clase tonal **A**. Como todas las notas cumplen esta propiedad y están geométricamente espaciadas, todas se pueden escribir de la siguiente forma:

$$f_n = 2^{\frac{n}{12}} f_0, \quad n \in [-50, 40] \tag{2.1.1}$$

donde n es un entero y el rango elegido es el de las notas más comunes utilizadas. Estamos hablando de frecuencias entre 24 y 4440 Hz (un rango de unas 7 octavas). Se dice que dos notas contiguas están separadas por un *semitono* ($f_{n+1} = 2^{1/12} f_n$).

La notación de dichas notas se complica un poco debido a que se realizan de forma un poco subjetiva dependiendo del tipo de escala que se esté usando. De forma general, el consenso es que hay siete clases tonales principales llamadas **la, si, do, re, mi, fa, sol** o en notación americana **A, B, C, D, E, F, G** cuyos intervalos son {2, 1, 2, 2, 1, 2} (en semitonos). Las cinco clases tonales restantes se obtienen sumando (#) o restando (b) semitonos a estas siete clases, es decir, si f_A es la frecuencia de la nota **A**, la frecuencia de la nota **B** es $f_B = 2^{2/12} f_A$ y por lo tanto a la nota intermedia de frecuencia $f = 2^{1/12} f_A = 2^{-1/12} f_B$ se le llama **la sostenido** ($\mathbf{A}^{\#}$) o **si bemol** ($\mathbf{B}^{b}$). Sin embargo, cuando un instrumento toca una nota cualquiera, no sólo produce esa frecuencia sino también otras (generalmente sus múltiplos entre otras). Este espectro de frecuencias que produce un instrumento es lo que se suele llamar el *timbre* y es lo que nos permite distinguir el instrumento que está produciendo el sonido. Por lo tanto, el espectro frecuencial de una canción no sólo indica las notas que se están tocando, sino también los instrumentos que están sonando.

Como se verá en las siguientes secciones, la gran mayoría de rasgos que pueden extraerse de una canción se obtienen a partir del espectro de frecuencias, que a su vez, podemos obtener utilizando la transformada de Fourier.

2.2. Transformada de Fourier

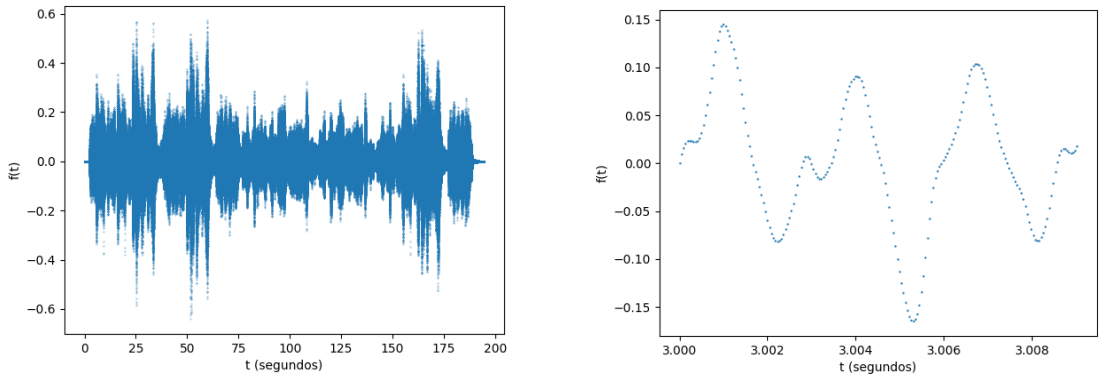
Desde una perspectiva matemática, una canción no es más que una señal, es decir, se trata de una función dependiente del tiempo $f(t)$ como la de la Figura 2.1, en principio continua. Muchas veces es más útil trabajar en el dominio frecuencial que en el temporal, sobre todo cuando son funciones aproximadamente sinusoidales como la de la Figura 2.1b. Para ello, se aplica la transformada de Fourier definida para una función continua como:

$$\mathcal{F}\{f(t)\} = \hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{2.2.1}$$

y cuya transformada inversa es:

$$\mathcal{F}^{-1}\{\hat{f}(\omega)\} = f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega \tag{2.2.2}$$

Sin embargo, como se puede observar en la Figura 2.1b, en realidad las señales con las que trabajamos no son continuas sino discretas, debido a que nuestros instrumentros de medida no son capaces de medir de forma continua sino que toman muestras cada cierto tiempo, llamado *periodo de muestreo* (T_s). En el caso de la Figura 2.1, se está usando un periodo de muestreo de aproximadamente 45.35 μs o de forma equivalente, una



(a) Señal completa. (b) Pequeño fragmento de la señal.

Figura 2.1: Tercer movimiento de la Sonata de Piano en do menor de Schubert.

frecuencia de muestreo (f_s) de 22050 Hz. El equivalente de la transformada de Fourier para señales discretas es la DTFT (discrete time Fourier transform). Sin embargo, como siempre utilizaremos un número finito de muestras (la canción tiene una duración finita), lo que utilizamos es en realidad la DFT (discrete Fourier transform) y su inversa, que vienen dadas por:

$$\hat{u}_k = \sum_{j=1}^{N} u_j e^{-ik\frac{2\pi}{N}j}, \quad k = -N/2+1, \ldots, N/2 \tag{2.2.3}$$

$$u_j = \frac{1}{N} \sum_{k=-N/2+1}^{N/2} \hat{u}_k e^{ik\frac{2\pi}{N}j}, \quad j = 1, \ldots, N \tag{2.2.4}$$

donde $u_j = u(T_s j)$ son las muestras tomadas y N el número de muestras. Además, la DFT se puede calcular usando un algoritmo extremadamente eficiente llamado FFT (fast Fourier transform). La DFT descompone la señal en una suma de señales sinusoidales de frecuencias k/N (kf_s/N el equivalente continuo) y amplitud $\hat{u}_k$. Pero esta descomposición se hace teniendo en cuenta todo el dominio temporal, obteniendo unas características globales de la señal. En el caso de una canción, esto puede resultar problemático, dado que una canción está compuesta por distintos pasajes y frases que pueden llegar a diferenciarse bastante espectralmente. Esto implica que al realizar la DFT estamos perdiendo ciertos detalles locales. Para remediar este problema podemos utilizar la STFT (short time Fourier transform). Para calcular la STFT lo que hacemos es calcular la transformada de un trozo de la función ($f(t)$) dado por una *función ventana* ($g(t)$) y desplazando esta ventana (un tiempo τ), de forma que calculamos la transformada de la función $f(t)g(t-\tau)$. En tiempo discreto, esto se puede escribir de la siguiente forma [5]:

$$\hat{u}_{k,m} = \sum_{j=1}^{N} u_{j+mH} w_j e^{-ik\frac{2\pi}{N}j} \tag{2.2.5}$$

donde w_j son las muestras de la función ventana (de tamaño M) y H el tamaño de salto ($\tau \to mT_sH$). Sin embargo, esta transformada presenta un problema importante en cuanto a la resolución frecuencial. Debido a la propiedad de escalado de la transformada de Fourier (si $g(t) = f(at)$ entonces $\hat{g}(\omega) = \hat{f}(\omega/a)/|a|$), reducir el tamaño de la ventana ($a' < a$) implica un ensanchamiento en la transformada, es decir, aumenta la distancia

entre las frecuencias que reconocemos o lo que es equivalente, la resolución frecuencial disminuye. Pero al mismo tiempo, tenemos un menor rango temporal en el cual identificamos dichas frecuencias lo que implica que la resolución temporal aumenta. De forma simétrica, al aumentar el tamaño de la ventana obtenemos una menor resolución temporal pero mayor resolución frecuencial. Este compromiso entre resoluciones es un factor importante a tener en cuenta al aplicar la STFT dependiendo de la tarea a realizar. Obviamente, la función ventana que utilicemos también es relevante para las frecuencias obtenidas. Las ventanas más usadas son Hamming, Hann y Blackman. Cada trozo de señal analizado al aplicar una de estas ventanas recibe el nombre de frame (que podríamos traducir como marco o fragmento).

Como se ha dicho en la sección anterior, en la música occidental las frecuencias de las notas utilizadas vienen dadas por la fórmula 2.1.1. Esto significa que lo que nos interesa es tener una buena resolución alrededor de dichas frecuencias. Esto se puede conseguir utilizando funciones ventana con un tamaño variable (N_k) en función de la frecuencia central, dando lugar a otra variante de la transformada llamada *Constant Q Transform* (CQT) [6]. Reordenando y renombrando algunas variables en la fórmula de la STFT (2.2.5), obtenemos la fórmula de la CQT:

$$\hat{u}_{k,m} = \frac{1}{N_k} \sum_{j=m-\lfloor N_k/2 \rfloor}^{m+\lfloor N_k/2 \rfloor} u_j \; w\left(\frac{j-m+N_k/2}{N_k}\right) e^{i2\pi \frac{k}{N}(j-m+N_k/2)} \tag{2.2.6}$$

donde el término $1/N_k$ se utiliza como factor de normalización. De forma similar a la STFT, el dominio frecuencial se divide en varios trozos de forma que cada trozo está centrado en la frecuencia $f_k = kf_s/N$. A continuación, se define una cantidad llamada *factor de calidad* (Q_k) para cada trozo, dado por [6]:

$$Q_k = \frac{f_k}{\Delta f_k} = \frac{N_k f_k}{\Delta w f_s} \tag{2.2.7}$$

donde Δf_k es el ancho de banda de todo lo que multiplica a u_j en la fórmula 2.2.6 (los trozos) y Δw es el ancho de banda de la función ventana. Como podemos intuir del nombre de esta transformada, la idea es utilizar un factor de calidad constante. Esto se debe, de nuevo, a la definición de las frecuencias de las notas 2.1.1. Si se divide cada octava en B trozos, entonces cada frecuencia central viene dada por $f_k = 2^{k/B} f_0$, y por lo tanto, el tamaño de cada trozo es $N_k = f_s/(f_{k+1} - f_k) = f_s/(2^{1/B} - 1)f_k$. Por lo que el factor de calidad es:

$$Q = \frac{1}{\Delta w(2^{1/B} - 1)} \tag{2.2.8}$$

Lo más común para la identificación de notas del TET es que B sea un múltiplo de 12. En el caso de la librería *librosa* [3], por defecto se utilizan $B = 36$ trozos.

2.3. Rasgos Representativos

Para la tarea de clasificación propuesta, se van a considerar 6 tipos distintos de rasgos, formando un total de 36 parámetros que servirán para identificar una canción. Estos rasgos son el *tempo*, *zero crossings*, *spectral centroid*, *spectral roll-off*, *MFCC* y el *cromagrama*.

Zero Crossings

Este rasgo mide el número de veces que la señal cruza el cero. Por ejemplo, en la Figura 2.1b se cruza el cero un total de 9 veces, y en el caso de la canción completa (2.1a), 209581 veces. Este parámetro es útil porque de forma muy sencilla no sólo refleja la velocidad de la canción, sino también el conjunto de frecuencias involucradas, dado que a mayor frecuencia, mayor será este número.

La librería *librosa* proporciona una función para calcular una lista con el número de zero crossings de cada frame. Para obtener un único parámetro con el número total de veces que se cruza el cero en una canción se suman los elementos de dicha lista.

Spectral Centroid

Como su nombre indica, este parámetro es el "centro de masa"de las frecuencias. En lugar de una masa, lo que se usa para promediar las frecuencias es la magnitud de la amplitud compleja $\hat{u}_k$:

$$C = \frac{\sum_k |\hat{u}_k| f_k}{\sum_k |\hat{u}_k|} \tag{2.3.1}$$

De nuevo, *librosa* proporciona una función para calcular el promedio espectral de cada frame. En este caso, se calcula el valor medio de todos los frames para obtener un único valor representativo del promedio espectral de una canción. Este parámetro es útil porque se asocia a una medida especial del timbre de la canción [7].

Spectral Roll-off

Este parámetro es la frecuencia de corte para la cual todas las frecuencias inferiores suman un cierto porcentaje de la energía total. Usando el teorema de Parseval, la densidad espectral de energía se define como $D_k = |\hat{u}_k|^2$ y por lo tanto, el spectral roll-off es la frecuencia f_r para la cual se cumple:

$$100 \cdot \frac{\sum_{k \leq r} |\hat{u}_k|^2}{\sum_k |\hat{u}_k|^2} = X\,\% \tag{2.3.2}$$

Esta cantidad nos indica en qué frecuencias se encuentra distribuida la mayor parte de la energía, y por lo tanto, sirve como indicador de cuáles son las frecuencias más importantes de la canción. Como con el resto de rasgos, la librería *librosa* proporciona una función para calcular el spectral roll-off en cada frame de la canción. Se utilizará el valor por defecto de 85 % y, al igual que con el promedio espectral, se calculará un único valor de spectral roll-off como el valor medio para todos los frames.

MFCC

Los Mel-Frequency Cepstral Coefficients (MFCC) fueron desarrollados originalmente para el reconocimiento automático del habla, debido a que estos coeficientes capturan las propiedades del timbre de la señal (vocales, consonantes, ...). Sin embargo, han demostrado ser muy útiles para capturar también las propiedades de una canción. Estos coeficientes se calculan usando una escala logarítmica y aplicando una serie de filtros triangulares a la señal [8]. La librería *librosa* proporciona una función para calcular estos

coeficientes para cada frame. Se usará el número predeterminado de coeficientes (20) y se calcula la media para todos los frames para obtener 20 coeficientes globales para cada canción.

Cromagrama

El cromagrama es una lista de doce números normalizados que nos indican la distribución de las frecuencias de la canción según las clases tonales, es decir, es una cuenta de la cantidad de notas tocadas en cada clase tonal. La librería *librosa* dispone de una función que calcula el cromagrama en cada frame usando la transformada CQT (usando por defecto 36 trozos por octava). Nuevamente, se calcula la media para todos los frames para obtener doce parámetros por canción. A pesar de que deja de estar normalizado, sigue siendo representativo de la canción y el cromagrama promedio además puede estandarizarse posteriormente.

Tempo

El tempo de una canción se refiere a su velocidad. Una canción occidental utiliza una subdivisión de cada compás en *tiempos*. Una subdivisión $\frac{2}{4}$ indica que cada compás tiene dos tiempos, o equivalentemente, que caben dos notas de valor uno (una *negra*). El tempo nos indica precisamente la duración de un tiempo; por ejemplo, el tiempo en segundos que dura una nota negra. Generalmente, el tempo se indica al principio de una canción en BPM (beats per minute) que significa cuantos tiempos caben en un minuto (la duración en segundos de un tiempo sería 60/BPM).

El cálculo del tempo de una canción es un problema más complicado de lo que pueda parecer. Cuando un instrumento toca una nota, se distinguen dos partes distintas: el *ataque* y el *transitorio*. El ataque se refiere al comienzo de la nota, en el cual ocurre un aumento brusco en la amplitud. Por lo tanto, para calcular el tempo es necesario detectar estos ataques. El problema es que esto se complica bastante cuando hay varios instrumentos sonando simultáneamente. Dado que este problema no es muy relevante para el objetivo de este trabajo, de forma muy resumida, lo que se hace es aplicar una compresión logarítimica al espectro frecuencial y luego se calcula la derivada temporal de este espectro comprimido para poder aplicar otro tipo de funciones que sirven para disminuir fluctuaciones mientras se mejora la estructura de los picos y así poder detectar los ataques [1]. De nuevo, la librería *librosa* proporciona una función para calcular el tempo de una señal.

[1] Para una explicación mucho más detallada, consultar el capítulo 6 de [5].

Capítulo 3

Técnicas de Aprendizaje Automático

En el capítulo anterior se han descrito los distintos rasgos que se utilizarán para representar una canción. De cara a la tarea de clasificación de compositores, esto implicaría que una canción nueva cuyos rasgos sean muy parecidos a los de las canciones de un compositor dado, es muy probable que haya sido compuesta por dicho compositor. La cuestión es ahora cómo determinar que los rasgos de una canción se parecen a los de otra, y la forma de usar esta información de similitud para predecir el compositor. Esto se puede conseguir utilizando algoritmos de *machine learning*. En este capítulo trataremos de explicar cómo funcionan las técnicas de aprendizaje automático de forma general y más detalladamente cómo funcionan algunos algoritmos en concreto con los que se han obtenido mejores resultados. Todos estos algoritmos están implementados en la librería *Scikit-learn* [4], que es la que se ha utilizado para obtener todos los resultados de este trabajo, mientras que las figuras se han obtenido usando los módulos *Matplotlib* [9] y *Seaborn* [10].

3.1. Introducción

Los tres elementos más importantes de un método de aprendizaje automático son la *representación*, las *etiquetas* y el *modelo*. Llamamos representación al conjunto de parámetros o rasgos utilizados para indentificar los objetos. Las representaciones se calculan a partir de una base de datos con representaciones a bajo nivel de un gran número de objetos (en este caso, señales con temas de música clásica). Las etiquetas son las respuestas, el valor real del parámetro que queremos predecir. En este caso, dado que se desea predecir los compositores, las etiquetas son nombres de compositores (Mozart, Bach, Beethoven, ...). El tercer elemento es, el modelo matemático que permitirá identificar regularidades en las características, y a partir de dichas regularidades, llevar a cabo la tarea de clasificación o predicción. Existen dos tipos principales de aprendizaje: aprendizaje *supervisado* y aprendizaje *no supervisado*.

Aprendizaje no Supervisado

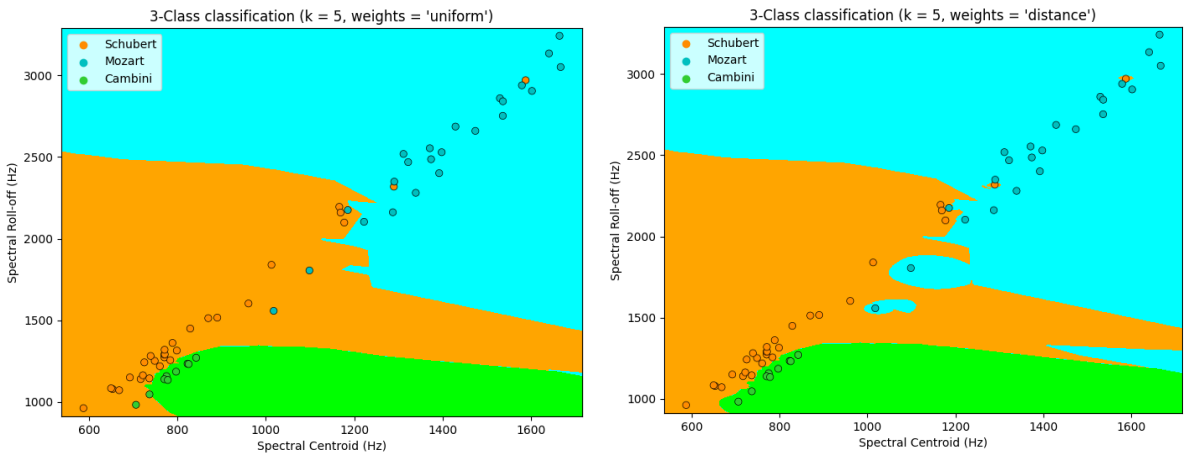
Este tipo de aprendizaje no requiere de etiquetas, los propios modelos se encargan de identificar las distintas agrupaciones presentes en los datos. Esto es muy útil sobre todo cuando se dispone de una gran cantidad de datos, de forma que no es necesario el etiquetado manual de cada objeto. Estos algoritmos se utilizan principalmente en sistemas de recomendación.

Aprendizaje Supervisado

Este tipo de aprendizaje es el más usado debido a su gran flexibilidad y rendimiento. En el aprendizaje supervisado, cada objeto está etiquetado con la clase a la que pertenece o con el valor de la variable que se desea predecir. De forma general, los algoritmos de aprendizaje supervisado utilizan alguna función matemática para medir la distancia entre el objeto a predecir y el resto de objetos conocidos (y correctamente etiquetados) y otra función basada en algún modelo estadístico con el cual predicen la etiqueta. En este trabajo, los métodos son todos de aprendizaje supervisado. En los siguientes apartados, se explica con detalle cada uno de ellos.

3.2. Nearest Neighbors

La idea básica de este método es muy sencilla: se considera un espacio de dimensión N, donde N es el número de rasgos extraídos de cada objeto, y se representa cada objeto como un punto en este espacio. De esta forma, cuando queramos predecir la etiqueta de un nuevo objeto, sólo tenemos que representarlo en este espacio y estudiar la distancia de este punto al resto de puntos. Hay principalmente dos formas de estudiar la distancia respecto al resto de puntos: considerando un número constante (k) de puntos próximos (los k puntos más próximos) o una esfera en N dimensiones de radio constante (r) y los puntos en su interior. En la Figura 3.1 se presenta un ejemplo visual del modelo KNN (k-nearest neighbors). En este caso se usan sólo dos rasgos (spectral centroid y spectral roll-off) para poder representarlo en dos dimensiones. Además, comparando ambas imágenes también se pueden observar los efectos de cambiar la forma en la que se ponderan los puntos. En la Figura 3.1a se usa un peso uniforme, es decir, todos los puntos se ponderan de igual forma. Sin embargo, en la Figura 3.1b se usa una ponderación según la distancia, es decir, los puntos más cercanos tienen más peso que los más lejanos.



(a) (b)

Figura 3.1: Ejemplo del modelo KNN utilizando 3 compositores del dataset MusicNet. Los puntos dibujados son las canciones utilizadas para entrenar el modelo y las zonas de colores indican qué compositor predice el modelo si la canción se encuentra en la zona.

Como cabe esperar, cambiar k (el número de puntos próximos a tener en cuenta) cambia la forma de estas zonas. Otro concepto importante en el modelo KNN es el de

métrica. Dado que el algorítmo se basa en la distancia entre los puntos, cambiar la forma en la que se calcula la distancia cambia completamente el resultado obtenido. En la Figura 3.1 se ha utilizado la métrica por defecto y la más utilizada: la *euclidea* ($d(\mathbf{x}, \mathbf{y}) = \sum_i \sqrt{(\mathbf{x}_i - \mathbf{y}_i)^2}$). Otras métricas bastante conocidas utilizadas en este modelo son:

- *manhattan*: $d(\mathbf{x}, \mathbf{y}) = \sum_i |(\mathbf{x}_i - \mathbf{y}_i)|$
- *chebyshev*: $d(\mathbf{x}, \mathbf{y}) = max(\sum_i |(\mathbf{x}_i - \mathbf{y}_i)|)$
- *canberra*: $d(\mathbf{x}, \mathbf{y}) = \sum_i |(\mathbf{x}_i - \mathbf{y}_i)|/(|\mathbf{x}_i| + |\mathbf{y}_i|)$
- *braycurtis*: $d(\mathbf{x}, \mathbf{y}) = \sum_i |(\mathbf{x}_i - \mathbf{y}_i)|/(\sum_j |\mathbf{x}_j| + \sum_k |\mathbf{y}_k|)$

Sus efectos se pueden observar en la Figura 3.2.

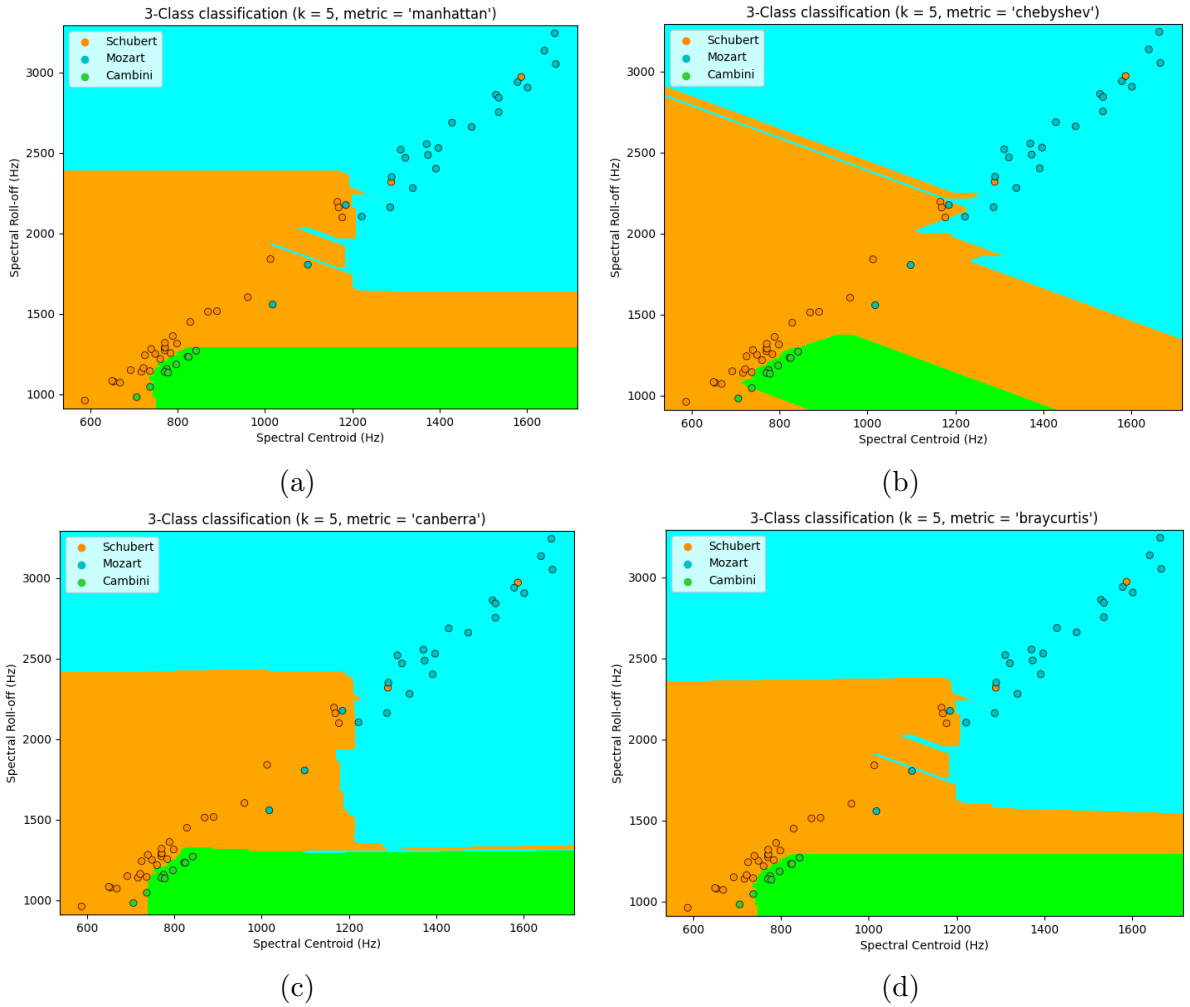


Figura 3.2: Ejemplo del efecto de la métrica en el modelo KNN (pesos uniformes). Los puntos dibujados son las canciones utilizadas para entrenar el modelo y las zonas de colores indican qué compositor predice el modelo si la canción se encuentra en la zona.

Observando la Figura 3.2 puede parecer que el cambio de la métrica no produce cambios significativos pero eso se debe a que la dimensión es pequeña, se tienen pocos puntos y pocas clases. En la tarea real se tienen miles de canciones separadas en diez o más compositores y de cada una se extraen 36 parámetros (36 dimensiones). Según pruebas

realizadas, la métrica puede mejorar la eficacia en hasta un 20 % y puede mejorar también la eficiencia del algoritmo dado que algunas métricas son capaces de obtener predicciones similares usando un k menor.

3.3. Support Vector Machines

La idea de este método es obtener representaciones de los objetos en un espacio de gran dimensión y luego obtener distintos hiperplanos que puedan separar las distintas clases [11], [12]. Para obtener un espacio de mayor dimensión lo que se hace es aplicar una función ($\phi : U \to V$) a los datos de entrenamiento ($\mathbf{x}$, y): $\mathbf{z} = \phi(\mathbf{x})$. Empecemos por el caso más simple: clasificación binaria. Se desea obtener un hiperplano que separe de la mejor manera los datos de cada clase. Un hiperplano general se puede escribir de la siguiente forma:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{3.3.1}$$

donde $\mathbf{w}$ es el vector normal al hiperplano y b una constante. La regla de clasificación que obtenemos de este hiperplano es:

$$G(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{z} + b) \tag{3.3.2}$$

donde $G(\mathbf{x})$ es la llamada *función de decisión* e indica la clase a la que pertenece el punto $\mathbf{x}$: 1 si está a un lado de la frontera y -1 si está al otro lado. En la Figura 3.3 se muestra un ejemplo. Como puede verse en la figura, no todos los puntos se pueden separar por este margen y ξ_i^* es la distancia a los respectivos márgenes de sus clases, es decir, el error que se comete con esta separación.

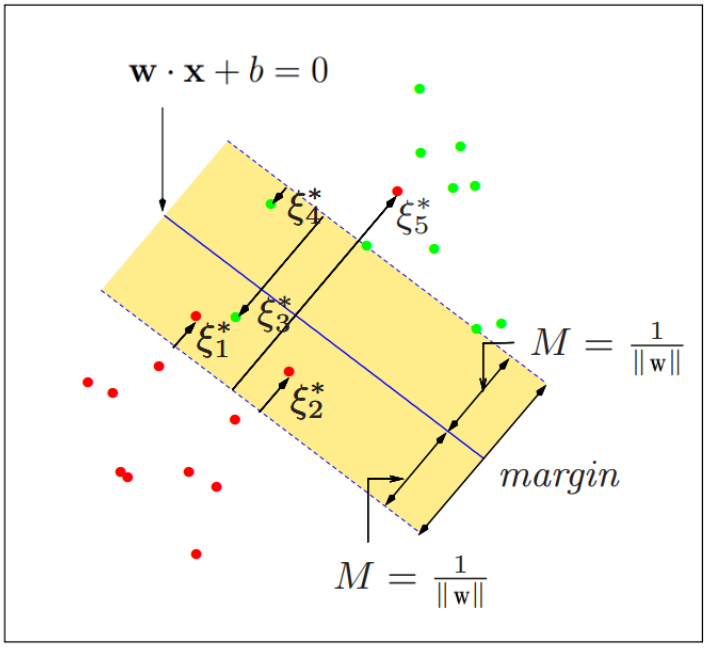


Figura 3.3: Ejemplo 2D del SVM lineal. [12] pg.418.

Se puede demostrar, minimizando funciones lagrangianas [1], que el hiperplano óptimo dado por $\mathbf{w}$ se puede escribir en función de un número reducido de puntos de entrenamiento llamados *support vectors* (veectores soporte). Generalmente esto se escribe de la siguiente forma:

$$\mathbf{w} = \sum_{i \in SV} y_i \alpha_i \mathbf{z}_i \tag{3.3.3}$$

donde i recorre los *support vectors*, los coeficientes α_i se calculan a partir de las ecuaciones lagranginas e $y_i = \{-1, 1\}$ es la clase a la que pertenece el punto de entrenamiento $\mathbf{z}_i$. Si nos fijamos en las ecuaciones 3.3.1 y 3.3.2, utilizamos el producto escalar usual para estas definiciones, pero esto no tiene por qué ser así. La ecuación del hiperplano y por lo tanto, de la función de decisión, se puede generalizar para cualquier tipo de espacio con un producto escalar especial $\langle \cdot, \cdot \rangle$:

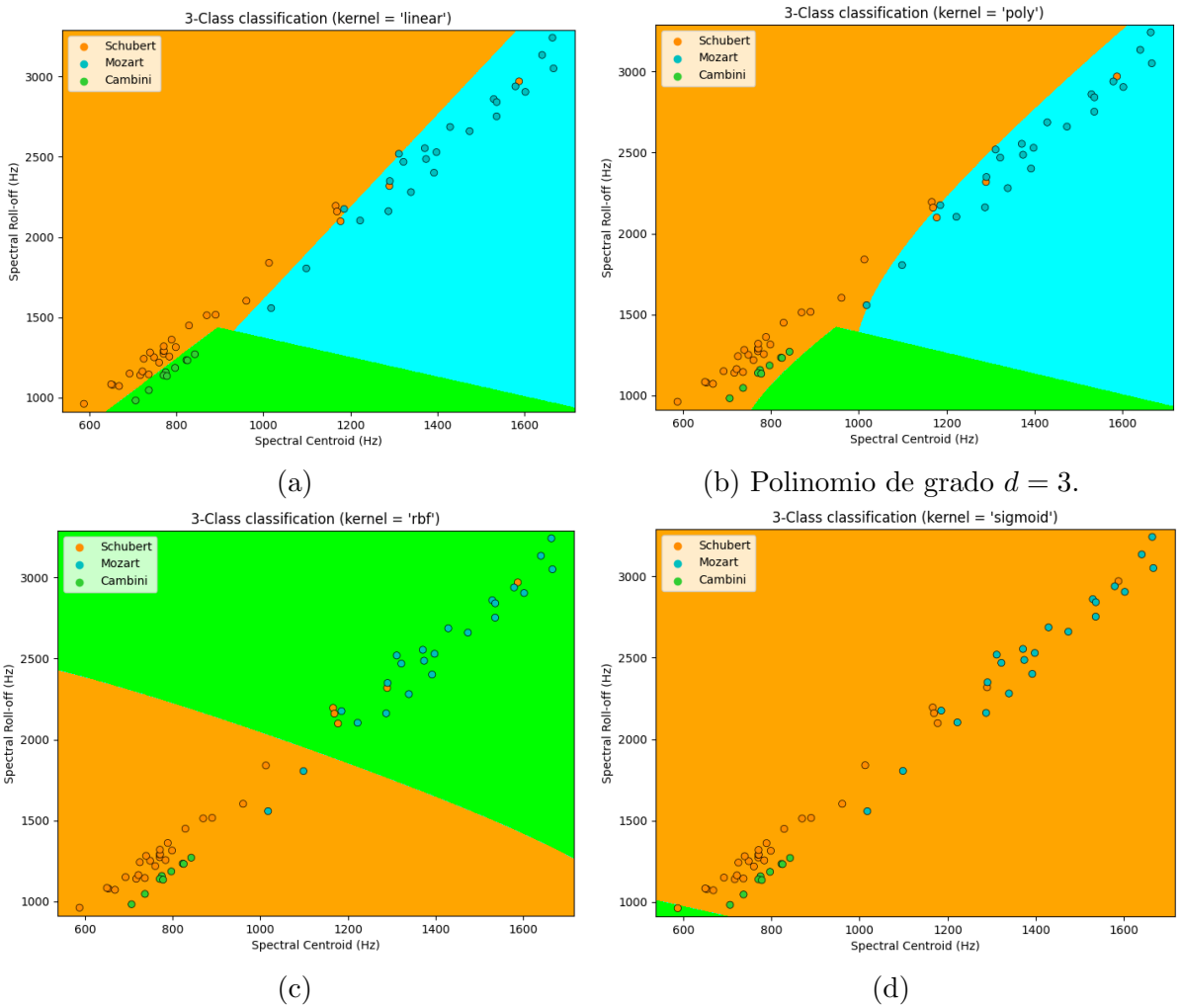
$$G(\mathbf{x}) = \text{sign}\left(\sum_{i \in SV} y_i \alpha_i \langle \mathbf{z}_i, \mathbf{z} \rangle + b \right) = \text{sign}\left(\sum_{i \in SV} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b \right) \tag{3.3.4}$$

donde $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ se le llama *kernel*. Existen varias funciones kernel diseñadas para distintos tipos de datos. Las más utilizadas y las que proporciona la librería *Scikit-learn* son:

- *lineal*: $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$
- *polinomial*: $k(\mathbf{x}, \mathbf{x}') = (\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + r)^d$
- *rbf*: $k(\mathbf{x}, \mathbf{x}') = e^{-\gamma |\mathbf{x} - \mathbf{x}'|^2}$
- *sigmoide*: $k(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + r)$

Como puede verse en la Figura 3.4, utilizando sólo dos rasgos, las funciones *rbf* y *sigmoide* no dan buenos resultados, pero como se verá en el próximo capítulo, en realidad el kernel *rbf* funciona muy bien para la tarea propuesta. La Figura 3.4 no es del todo representativa de cada kernel, dado que tienen varios parámetros que se pueden afinar para producir un mejor resultado para el conjunto de datos considerado. Un ejemplo claro de esto se puede ver en la Figura 3.5, donde simplemente cambiando el grado del kernel polinomial, observamos una gran mejora en la predicción. Esto, por supuesto, conlleva un mayor gasto computacional y para mayores cantidades de rasgos, muchas veces la mejora no es suficientemente significativa y a veces incluso el modelo empeora.

[1] El proceso completo se explica en el capítulo 12 de [12].



(a) (b) Polinomio de grado $d = 3$.

(c) (d)

Figura 3.4: Ejemplo del efecto de las funciones kernel en el modelo SVM. Los puntos dibujados son las canciones utilizadas para entrenar el modelo y las zonas de colores indican qué compositor predice el modelo si la canción se encuentra en la zona.

Un aspecto importante pasado por alto es la forma en la que se realiza una clasificación multiclase. El modelo matemático de una SVM (ecuación 3.3.4) está definido únicamente para un problema binario (dos clases). Existen principalmente dos formas de extender modelos binarios a multiclase: "*one-versus-one*"(OVO) o "*one-versus-rest*"(OVR). En OVO lo que se hace es crear un modelo para cada posible pareja de etiquetas (eso son $\binom{N}{2}$ modelos) y la nueva canción se clasifica usando cada de uno de los modelos de forma que la clase con mayor número de predicciones entre todos los modelos es la predicción final. Por otra parte, en OVR lo que se hace es crear un modelo que compare una única clase con el resto de clases (eso son simplemente N modelos) y gana el modelo que minimice alguna métrica que represente el error. En el caso de las SVM, la menor desviación de puntos del hiperplano. En el caso de las Figuras 3.4 y 3.5 se ha usado el modelo OVR.

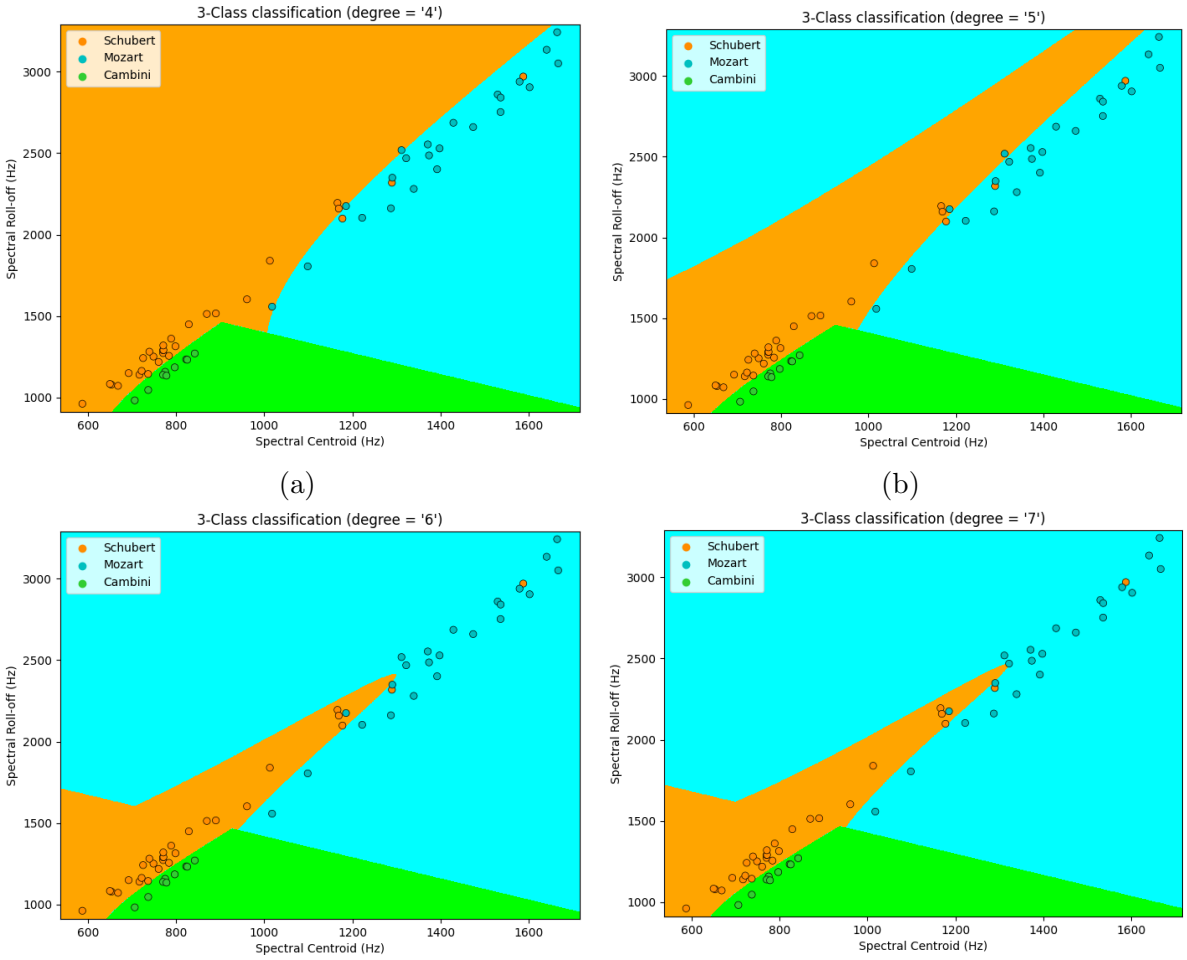


Figura 3.5: Ejemplo del efecto del grado del kernel polinomial en el modelo SVM. Los puntos dibujados son las canciones utilizadas para entrenar el modelo y las zonas de colores indican qué compositor predice el modelo si la canción se encuentra en la zona.

3.4. Gaussian Processes

Los procesos Gaussianos son modelos completamente probabilísticos, usualmente llamados modelos Bayesianos debido a su extenso uso del teorema de Bayes [2]. Para entender mejor cómo funciona este método, empecemos considerando un problema de regresión. Se dispone de una serie de muestras $(\mathbf{x}_i, y_i)$, en lo que llamamos conjunto de entrenamiento $(X, \mathbf{y})$, y se desea descubrir la función que relaciona estas muestras $f(\mathbf{x}_i) = y_i$. Supongamos que la función viene dada por $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$, donde $\mathbf{w}$ es un vector de pesos y que los valores observados (y) difieren de esta función por un ruido dado por una distribución Gaussiana de media nula y varianza σ^2: $y = f(\mathbf{x}) + \mathcal{N}(0, \sigma^2)$. Suponiendo independencia estadística en las muestras, la probabilidad de obtener las etiquetas $\mathbf{y}$ a partir de los datos de entrenamiento X y los pesos $\mathbf{w}$ es:

[2]Esta sección expone de forma extremadamente resumida los contenidos relevantes de [13] y la sección 6.4 de [14].

$$p(\mathbf{y}|X,\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_i \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y_i-\mathbf{w}\cdot\mathbf{x}_i)^2}{2\sigma^2}\right) = \mathcal{N}(X\mathbf{w},\sigma^2 I) \quad (3.4.1)$$

Ahora recordemos el teorema de Bayes, que viene dado por:

$$p(\mathbf{w}|\mathbf{y},X) = \frac{p(\mathbf{y}|X,\mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|X)} \quad (3.4.2)$$

donde $p(\mathbf{w})$ es la distribución previa de los pesos, que podemos suponer otra Gaussiana de media nula ($\mathbf{w} \sim \mathcal{N}(0,\Sigma)$, Σ es la matriz de covarianza), y la probabilidad de $\mathbf{y}$ dado X se puede calcular de la siguiente forma

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X,\mathbf{w})p(\mathbf{w})d\mathbf{w} \quad (3.4.3)$$

Con todo esto, resulta que la distribución de probabilidad de los pesos dado el conjunto de entrenamiento ($p(\mathbf{w}|\mathbf{y},X)$) se puede expresar mediante otra Gaussiana. Para hacer predicciones simplemente se realiza una media ponderada de las posibles funciones $f(\mathbf{x})$ utilizando la distribución obtenida ($p(\mathbf{w}|\mathbf{y},X)$), llegando nuevamente a otra Gaussiana ($f_*(y,\mathbf{x})$). Al igual que las SVM, muchas veces es útil aumentar la dimensión de los datos de entrenamiento, de modo que el proceso anterior se aplica a la variable $\mathbf{z} = \phi(\mathbf{x})$. Hacer esto lleva a la aparición de términos de la forma $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})\Sigma\phi(\mathbf{x}')$ en la Gaussiana que utilizamos para predecir. La notación $k(\mathbf{x},\mathbf{x}')$ no es una coincidencia ya que esta función se conoce como *función covariante* (dado que genera la matriz de covarianza del proceso Gaussiano) o, al igual que en las SVM, *kernel*.

El proceso Gaussiano descrito da como resultado una distribución de probabilidades ($f_*(y,\mathbf{x})$) para el valor y. Sin embargo, muchas veces se desea predecir nuevos puntos y por lo tanto, hay que seleccionar un valor para y. La forma óptima de realizar esta decisión es mediante una *función de pérdida* ($\mathcal{L}(y_{real},y)$). Esta función proporciona una puntuación que estima lo mala que es nuestra predicción y, pero para hacerlo se necesita conocer el valor real y_{real}, que en la práctica se desconece. Por lo tanto, lo que se utiliza realmente es la *pérdida esperada*, dada por la fórmula:

$$R_{\mathcal{L}}(y|\mathbf{x}) = \int \mathcal{L}(y',y)f_*(y',\mathbf{x})dy' \quad (3.4.4)$$

El valor óptimo para y será el que minimice esta función $R_{\mathcal{L}}$.

Ahora considerese una tarea de clasificación binaria dada por el conjunto de entrenamiento $S = \{(\mathbf{x}_i,y_i) \mid i = 1,\ldots,N,\ y_i \in \{-1,1\}\}$. Lo que se suele hacer es simplemente aplicar una función sigmoide $\sigma(x)$ al problema de regresión anterior, de forma que la probabilidad de que la clase de $\mathbf{x}$ sea $y = 1$ es:

$$p(y=1|\mathbf{x},\mathbf{w}) = \sigma(\mathbf{w}\cdot\mathbf{x}) \quad (3.4.5)$$

y, por lo tanto, la probabilidad de que la clase sea $y = -1$ es simplemente:

$$p(y=-1|\mathbf{x},\mathbf{w}) = 1 - p(y=1|\mathbf{x},\mathbf{w}) = 1 - \sigma(\mathbf{w}\cdot\mathbf{x}) \quad (3.4.6)$$

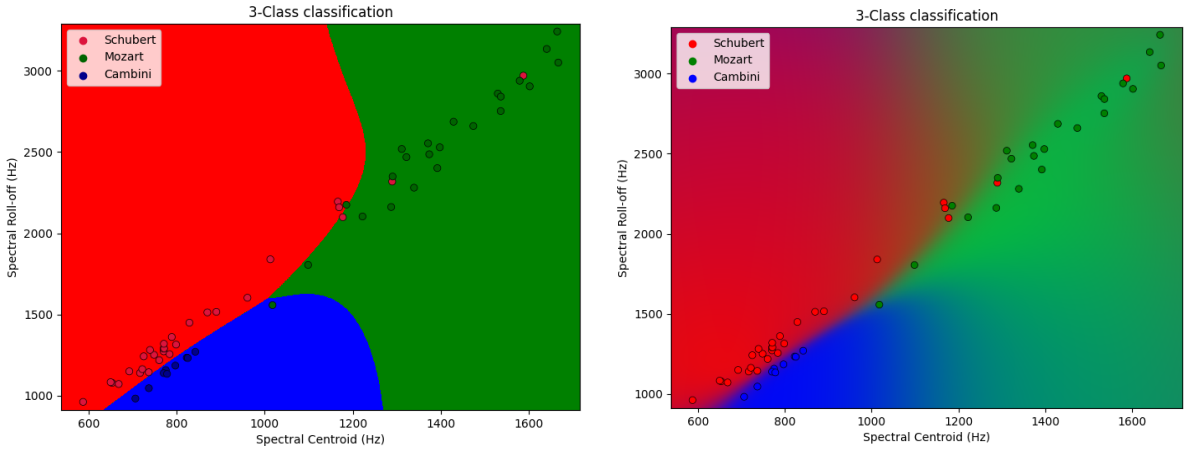
Una función sigmoide muy utilizada para este tipo de problemas es la función logística, dada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.4.7}$$

El proceso que se sigue es análogo al caso de regresión explicado. Utilizando el teorema de Bayes se calcula $p(\mathbf{w}|\mathbf{y}, X)$ y para obtener la distribución de la predicción se realiza la media ponderada

$$p(y = 1|\mathbf{x}, \mathbf{y}, X) = \int \sigma(\mathbf{w} \cdot \mathbf{x}) p(\mathbf{w}|\mathbf{y}, X) d\mathbf{w} \tag{3.4.8}$$

Finalmente, para obtener una predicción óptima de la clase y, se aplican otra vez los conceptos de la función de pérdida. Existe un único problema y es que, a pesar de que la distribución previa de $\mathbf{w}$ sea una Gaussiana, $p(\mathbf{w}|\mathbf{y}, X)$ no lo es. Eso, combinado con la función sigmoide que utilicemos, hace que la ecuación 3.4.8 generalmente no se pueda resolver de forma analítica. Una forma sencilla de resolver el problema podría ser evaluar la integral numéricamente pero lo que generalmente se hace es una aproximación de $p(\mathbf{w}|\mathbf{y}, X)$ mediante Gaussianas para que la integral sea analíticamente resoluble. A este método se le conoce como *aproximación de Laplace* [3] y es el método que utiliza la librería *Scikit-learn*. Al igual que con las SVM, se ha definido el proceso Gaussiano únicamente para una clasificación binaria. Esto se debe a que, al igual que con las SVM, lo más habitual es utilizar los métodos OVO y OVR para extender el proceso Gaussiano binario a una clasificación multiclase y este es precisamente el enfoque que utiliza la librería *Scikit-learn*. Se puede, sin embargo, utilizar una función logística múltiple o *softmax* para llevar a cabo directamente una clasificación multiclase. En la Figura 3.6 se comparan las probabilidades obtenidas con este modelo frente a las predicciones que se realizan al utilizar la función de pérdida.



(a) Zonas predichas de forma óptima mediante la función de pérdida.

(b) Mapa de colores según la distribución de probabilidad de cada clase.

Figura 3.6: Ejemplo del Gaussian Process Classifier (GPC) utilizando el kernel *rbf*. Los puntos dibujados son las canciones utilizadas para entrenar el modelo y las zonas de colores indican qué compositor predice el modelo si la canción se encuentra en la zona.

[3] Para más información sobre el proceso completo consultar la sección 3.4 de [13] y la sección 6.4.6 de [14].

3.5. Ensemble Methods: Voting Classifier

El término *Ensemble Methods* se refiere a un conjunto de métodos cuyo propósito es el de combinar las predicciones realizadas por varios modelos con el objetivo de mejorar la robustez y obtener una mayor generalización. Existen muchos tipos de ensembles creados con diferentes propósitos y utilizando distintos modelos para combinar las predicciones, tales como Bagging, Random Forests, AdaBoost, etc. Sin embargo, el método que mejor ha funcionado en pruebas preliminares es, casualmente, el más sencillo de todos: el Voting Classifier (VC). El VC tiene dos modos de "votación": *hard* y *soft*. El modo hard es exactamente la forma más sencilla que se nos pueda ocurrir: voto por mayoría, es decir, que la predicción sea la clase que más se ha predicho entre todos los modelos. El modo soft es un poco más sutil: se hace una media ponderada de las probabilidades proporcionadas por cada modelo y se toma como predicción la de mayor probabilidad resultante. Además, los pesos que se usan en la media ponderada se pueden ajustar manualmente para dar más importancia a las predicciones de algún modelo en concreto (más fiable). Si el peso del modelo i es w_i y la probabilidad de la clase j predicha por el modelo i es p_{ij}, entonces las probabilidades finales de cada clase son:

$$p_j = \frac{1}{N} \sum_{i}^{N} w_i \cdot p_{ij} \tag{3.5.1}$$

donde N es el número de modelos. En la Tabla 3.1 se muestra un ejemplo, donde se ha utilizado $w_i = 1 \ \forall i$.

Modelo	Clase 1	Clase 2	Clase 3
Modelo 1	$w_1 \cdot 0.44$	$w_1 \cdot 0.28$	$w_1 \cdot 0.28$
Modelo 2	$w_2 \cdot 0.55$	$w_2 \cdot 0.16$	$w_2 \cdot 0.29$
Modelo 3	$w_3 \cdot 0.37$	$w_3 \cdot 0.22$	$w_3 \cdot 0.31$
Resultado	0.45	0.22	0.29

Tabla 3.1: Ejemplo del modo de votación soft. En este caso se elegiría la clase 1 como predicción final.

Como se ha mencionado al principio del capítulo, los métodos explicados en secciones anteriores son los que mejor han funcionado para la tarea y conjunto de datos considerados, todos con un rendimiento parecido. El VC resulta ser especialmente útil cuando utilizamos modelos con rendimientos similares.

3.6. Transformaciones del Conjunto de Datos

No basta con extraer rasgos de las canciones para entrenar un modelo y luego predecir etiquetas. Hay que pensar cómo estos datos se van a utilizar dependiendo del modelo elegido. Para empezar, de cara a la tarea de predicción de compositores, las etiquetas son palabras, lo que se conoce como *strings* en cualquier lenguaje de programación de alto nivel. Esto es un problema bastante grande para la mayoría de modelos, como por ejemplo en el GPC, donde se usa el valor y no sólo para obtener las probabilidades sino también para predecir usando la función de pérdida. Por ello, lo primero que hay que hacer casi siempre al utilizar algoritmos de aprendizaje automático es aplicar una transformación adecuada a los datos.

La primera y más importante transformación, como se ha comentado, suele ser la conversión de las clases a números enteros (Schubert $\rightarrow$ 1, Mozart $\rightarrow$ 2, Cambini $\rightarrow$ 3, por ejemplo). Esto es fácil dado que la propia librería *Scikit-learn* proporciona una clase llamada *Label Encoder* que puede por sí sola cambiar todas las instancias de un compositor por un número entero y guardar esta asociación para poder recuperarlo más tarde si fuera necesario (sobre todo al predecir).

Otras transformaciones especialmente importantes son el escalado, la estandarización y el mapeado. Los escalados y mapeados sirven para cambiar el rango de los valores a unos más manejables, algo bastante importante dado que la mayoría de métodos funcionan mejor con valores pequeños y algunos incluso no funcionan con valores muy grandes. Algo muy parecido es la estandarización que se suele referir a la transformación $\mathbf{z} = (\mathbf{x} - \mu)/\sigma$, donde μ es el valor medio del rasgo $\mathbf{x}$ y σ su desviación estándar. Esta transformación es de las más utilizadas dado que muchos modelos funcionan bastante mal si los datos no se parecen a una distribución normal (una Gaussiana de media nula, por ejemplo). La librería *Scikit-learn* proporciona una clase llamada *Standard Scaler* con la cual se puede realizar esta transformación fácilmente.

Finalmente, la librería *Scikit-learn* proporciona otra clase especialmente útil llamada *Pipeline* con la cual es posible componer transformaciones ($(f \circ g)(x) = f(g(x))$) y aplicar el resultado directamente al modelo que se desee. Si se desea aplicar las dos transformaciones explicadas ($z = \mathrm{LE}(x)$ y $z = \mathrm{SS}(x)$) al modelo GPC, el *Pipeline* se encarga de hacer la operación $z = \mathrm{GPC}(\mathrm{SS}(\mathrm{LE}(x)))$.

Capítulo 4

Estudio y Comparación de los Resultados de Clasificación

En este capítulo se hará uso de todo lo explicado en capítulos anteriores para extraer los datos necesarios de ambos conjuntos de datos (MAESTRO [1] y MusicNet [2]), entrenar distintos modelos y finalmente, obtener resultados en forma de predicciones tanto determinísticas como probabilísticas. Como se ha mencionado en el capítulo anterior, se van a presentar y analizar resultados obtenidos usando únicamente cuatro métodos, los que mejor han funcionado en experimentos preliminares. Sin embargo, no son estos los únicos métodos que se han probado. Se han aplicado todos los métodos de entrenamiento supervisado para tareas de clasificación disponibles en la librería *Scikit-learn*, tales como *regresión lineal y logística*, *quadratic discriminant analysis*, *stochastic gradient descent*, *árboles de decisión* y *redes neuronales*, además de algún otro método que no se encuentra en la librería como *extreme gradient boosting*. Esto no significa que no exista ningún otro método que funcione mejor para nuestra tarea, sino que con el tiempo disponible no se han podido encontrar métodos tan eficaces como los cuatro métodos mencionados. Es posible que afinando los hiperparámetros de los modelos fuera posible mejorar los resultados, sin entrar a considerar muchos otros métodos que no se han probado.

4.1. Proceso de Extracción de Rasgos

Ambos conjuntos de datos disponen de un archivo de *metadatos* ("_metadata.csv" y "maestro-v3.0.0.csv"). Estos archivos contienen una tabla con todos los archivos de audio y la información asociada a cada uno de ellos (Figura 4.1). Esto es importante porque contienen las etiquetas de cada canción, es decir, los compositores.

Como se ha explicado en el capítulo 2, se ha aplicado la librería *librosa* para extraer los rasgos de cada canción. Dado que se desea probar distintos modelos y configuraciones, si tuviéramos que extraer todos estos datos cada vez que ejecutáramos el programa nunca terminaríamos. La forma eficiente de hacer esto es extraer los datos en forma de tabla y guardarlos en un archivo *.csv* justo como los archivos de metadatos. De esta forma, cada vez que queramos probar nuevos modelos sólo tenemos que leer el archivo. La forma más fácil de hacer esto en Python es por medio de la librería *pandas* [15]. Esta librería proporciona un objeto llamado *DataFrame* que combina las funcionalidades de tablas y arrays, y proporciona métodos para guardar y leer estas tablas en archivos *.csv*. Guardar los datos en un archivo es un paso completamente necesario dado que la extracción de rasgos tarda alrededor de un minuto por canción (más o menos, dependiendo de la duración y del ordenador), lo que nos da una estimación de unas 6 horas para MusicNet y 21

id	composer	composition	movement	ensemble	source	transcriber	catalog_name	seconds
1727	Schubert	Piano Quintet in A major	2. Andante	Piano Quintet	European Archive	http://tirolmusic.blogspot.com/	OP114	447
1728	Schubert	Piano Quintet in A major	3. Scherzo: Presto	Piano Quintet	European Archive	http://tirolmusic.blogspot.com/	OP114	251
1729	Schubert	Piano Quintet in A major	4. Andantino - Allegretto	Piano Quintet	European Archive	http://tirolmusic.blogspot.com/	OP114	444
1730	Schubert	Piano Quintet in A major	5. Allegro giusto	Piano Quintet	European Archive	http://tirolmusic.blogspot.com/	OP114	368
1733	Schubert	Piano Sonata in A major	2. Andantino	Solo Piano	Museopen	Segundo G. Yogore	D959	546
1734	Schubert	Piano Sonata in A major	3. Scherzo. Allegro vivace	Solo Piano	Museopen	Segundo G. Yogore	D959	325
1735	Schubert	Piano Sonata in A major	4. Rondo. Allegretto	Solo Piano	Museopen	Segundo G. Yogore	D959	714
1739	Schubert	Piano Trio in B-flat major	4. Rondo. Allegro vivace	Piano Trio	European Archive	harfesoft.de	OP99	490
1742	Schubert	String Quintet in C major	2. Adagio	Viola Quintet	European Archive	harfesoft.de	OP163	924
1749	Schubert	Piano Sonata in A minor	1. Moderato	Solo Piano	Museopen	Segundo G. Yogore	D845	696
1750	Schubert	Piano Sonata in A minor	2. Andante poco mosso	Solo Piano	Museopen	Segundo G. Yogore	D845	784
1751	Schubert	Piano Sonata in A minor	3. Scherzo. Allegro	Solo Piano	Museopen	Segundo G. Yogore	D845	475
1752	Schubert	Piano Sonata in A	4. Rondo. Allegro	Solo Piano	Museopen	Segundo G. Yogore	D845	362

(a) Metadatos de MusicNet.

canonical_composer	canonical_title	split	year	midi_filename	audio_filename	duration
Alban Berg	Sonata Op. 1	train	2018	2018/MIDI-Unprocessed_Chamber3_MID--AUDIO_10_R3_2018_wav--1.midi	2018/MIDI-Unprocessed_Chamber3_MID--AUDIO_10_R3_2018_wav--1.wav	698.661160312
Alban Berg	Sonata Op. 1	train	2008	2008/MIDI-Unprocessed_03_R2_2008_01-03_ORIG_MID--AUDIO_03_R2_2008_wav--2.midi	2008/MIDI-Unprocessed_03_R2_2008_01-03_ORIG_MID--AUDIO_03_R2_2008_wav--2.wav	759.51867125
Alban Berg	Sonata Op. 1	train	2017	2017/MIDI-Unprocessed_0to_PIANO0to9_RID--AUDIO-split_07-07-17_Piano-e_3-02_wav--3.midi	2017/MIDI-Unprocessed_0to_PIANO0to9_RID--AUDIO-split_07-07-17_Piano-e_3-02_wav--3.wav	484.649432068
Alexander Scriabin	24 Preludes Op. 11, No. 13-24	train	2004	2004/MIDI-Unprocessed_XP_21_R1_2004_01_ORIG_MID--AUDIO_21_R1_2004_01_Track01_wav.midi	2004/MIDI-Unprocessed_XP_21_R1_2004_01_ORIG_MID--AUDIO_21_R1_2004_01_Track01_wav.wav	872.6605680896
Alexander Scriabin	3 Etudes, Op. 65	validation	2006	2006/MIDI-Unprocessed_17_R1_2006_01-06_ORIG_MID--AUDIO_17_R1_2006_04_Track04_wav.midi	2006/MIDI-Unprocessed_17_R1_2006_01-06_ORIG_MID--AUDIO_17_R1_2006_04_Track04_wav.wav	397.6975608125
Alexander Scriabin	5 Preludes, Op.15	validation	2009	2009/MIDI-Unprocessed_07_R1_2009_04-05_ORIG_MID--AUDIO_07_R1_2009_07_R1_2009_04_BAV.midi	2009/MIDI-Unprocessed_07_R1_2009_04-05_ORIG_MID--AUDIO_07_R1_2009_07_R1_2009_04_BAV.wav	400.557829938
Alexander Scriabin	Entrag̀te, Op.63	test	2009	2009/MIDI-Unprocessed_11_R1_2009_09-09_ORIG_MID--AUDIO_11_R1_2009_11_R1_2009_07_BAV.midi	2009/MIDI-Unprocessed_11_R1_2009_09-09_ORIG_MID--AUDIO_11_R1_2009_11_R1_2009_07_BAV.wav	163.745829844
Alexander Scriabin	Etude Op. 2 No.1; Etudes Op. 8, Nos. 5, 11 and 12	train	2013	2013/ORIG-MIDI_03_7_8_13_Group__MID--AUDIO_19_R2_2013_wav--3.midi	2013/ORIG-MIDI_03_7_8_13_Group__MID--AUDIO_19_R2_2013_wav--3.wav	563.9043580625
Alexander Scriabin	Etude Op. 42, Nos. 4 & 5	test	2009	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MID--AUDIO_02_R1_2009_02_R1_2009_04_BAV.midi	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MID--AUDIO_02_R1_2009_02_R1_2009_04_BAV.wav	136.3153017197
Alexander Scriabin	Etude Op. 8, No. 11	validation	2009	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MID--AUDIO_02_R1_2009_05_R1_2009_05_BAV.midi	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MID--AUDIO_02_R1_2009_02_R1_2009_05_BAV.wav	167.085837188
Alexander Scriabin	Etude In D-Flat Major, Op. 8 No. 10	train	2011	2011/MIDI-Unprocessed_15_R1_2011_MID--AUDIO_R1-04_09_Track09_wav.midi	2011/MIDI-Unprocessed_15_R1_2011_MID--AUDIO_R1-04_09_Track09_wav.wav	102.007109531
Alexander Scriabin	Etudes from Op.8	train	2006	2006/MIDI-Unprocessed_19_R1_2006_01-07_ORIG_MID--AUDIO_19_R1_2006_04_Track04_wav.midi	2006/MIDI-Unprocessed_19_R1_2006_01-07_ORIG_MID--AUDIO_19_R1_2006_04_Track04_wav.wav	550.997045156
Alexander Scriabin	Fantasy Op. 28 in B Minor	train	2013	2013/ORIG-MIDI_02_7_10_13_Group_MID--AUDIO_11_R3_2013_wav--3.midi	2013/ORIG-MIDI_02_7_10_13_Group_MID--AUDIO_11_R3_2013_wav--3.wav	512.1642848464

(b) Metadatos de MAESTRO.

Figura 4.1: Archivos de metadatos de cada conjunto.

horas para MAESTRO (con un único núcleo trabajando).

Lo más importante al usar algoritmos de aprendizaje automático es la capacidad de generalización, es decir, se espera que el modelo funcione con un rendimiento similar al que se ha obtenido en los experimentos de test. El hecho de que MusicNet tan sólo disponga de 330 canciones supone un problema, por dos razones. La primera es que es un número bastante bajo de muestras con las que entrenar el modelo. Esto no significa que no se pueda obtener un buen modelo, pero de forma general, a mayor número de muestras de entrenamiento mejor será el modelo. La segunda razón es que para comprobar el rendimiento del modelo es necesario apartar unas cuantas canciones como parte del *conjunto de test*. El propio creador de MusicNet separa 10 canciones para realizar el test, pero esto es muy poco. Aunque el modelo obtuviera un 100 % de rendimiento, es decir, aunque predijera todas las canciones de test correctamente, no sería estadísticamente relevante. Incluso si se estableciera la separación usual de 80-20, lo cual dejaría 66 canciones para testear, seguiría siendo un número bastante bajo y además se tendrían menos canciones para entrenar. La forma más sencilla de resolver este problema es dividir cada canción en trozos más pequeños e interpretar cada trozo como una canción independiente.

Para empezar, los archivos de audio se leen mediante la función *load* de la librería *librosa*, que retorna una lista y con los valores de amplitud de la forma de onda del audio y la frecuencia de muestreo utilizada (sr). Utilizando estas dos variables podemos crear un generador que separe la canción en trozos de cualquier duración (véase el listado 4.1). El proceso es sencillo, dado que se conoce la cantidad de muestras de cada trozo (frecuencia de muestreo $\times$ duración) y Python tiene la funcionalidad de seleccionar trozos de listas. La única consideración importante es qué hacer con el último trozo, es decir, si una canción dura 200 segundos, pueden obtenerse 3 trozos de un minuto pero sobrarían 20 segundos de canción. Un trozo de 20 segundos no supone ningún problema, pero en casos extremos de que el trozo sobrante dure una décima de segundo, podemos tener problemas al aplicar la STFT o CQT, y por esta razón, las funciones de *librosa* exigen una mínima longitud para la lista y. Para garantizar un último trozo de longitud apropiada, se añade el parámetro m

que indica la mínima duración posible en forma de porcentaje con respecto a la duración deseada: un valor $m = 0.2$ implica un 20 % de la duración. Para trozos de 60 segundos implicaría que el último trozo debe de ser mayor que 12 segundos. Esta función se puede escribir en Python de la siguiente forma:

```
def gen_parts(y, sr, duration=60, m=0.2):
    samples = sr * duration
    n = int(len(y) / samples)
    for i in range(n):
        yield [i, y[samples * i: samples * (i+1)]]
    aux = y[samples * n:]
    if len(aux) >= samples * m:
        yield [n, aux]
```

Listing 4.1: Función generadora de trozos.

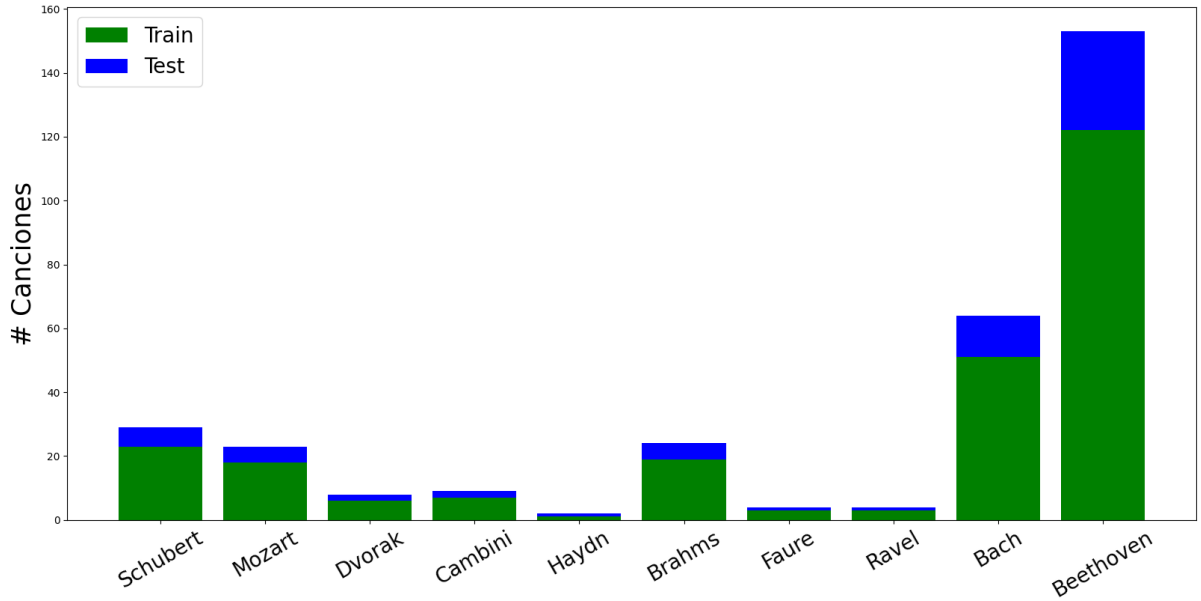
El único inconveniente de realizar este troceado de las canciones es que el proceso de extracción de rasgos tarda muchísimo más, hasta varios días. Por lo tanto, lo recomendable es emparejar este proceso con un procedimiento multihilo. En nuestro caso, esto se puede hacer de forma sencilla mediante la clase *Pool* de la librería estándar de Python llamada *multiprocessing*. Se aplica la función *starmap* para delegar automáticamente a varios hilos la extracción de rasgos de las partes generadas mediante la función mostrada en el listado 4.1, pasando de varios días de trabajo a unas pocas horas.

4.2. Distribución de Datos

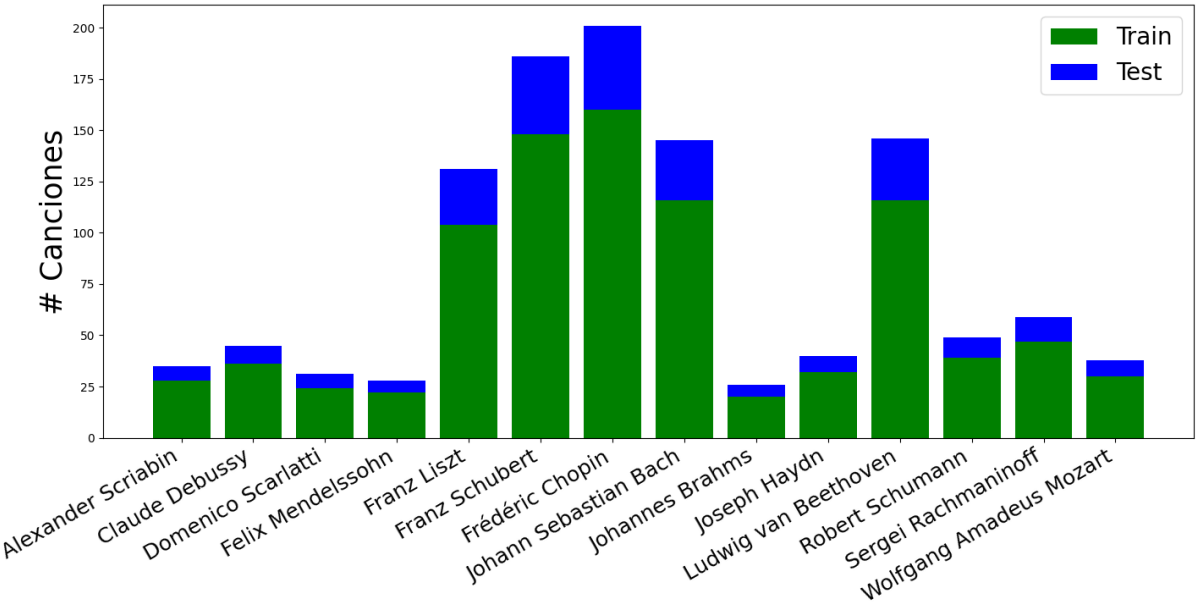
Lo primero que se hace necesario es una separación adecuada de los datos en un conjunto de entrenamiento y otro de test. Muchas veces se utiliza también un conjunto de *validación* con el cual afinar los hiperparámetros antes de hacer el test, pero no se ha utilizado en esta tarea. Un aspecto importante a tener en cuenta al hacer la separación es que interesa tener la misma representación de clases en ambos conjuntos, es decir, si la clase 1 tiene el doble de datos que la clase 2 en el conjunto de entrenamiento, que también tenga esa misma proporción en el conjunto de test. Esto se puede conseguir de forma bastante sencilla si se aplica la separación 80-20, por ejemplo, a los datos de cada clase por separado. Un ejemplo de esto lo podemos ver en la Figura 4.2. En el caso del dataset MAESTRO, hay varios compositores que tienen muy pocas canciones y sólo empeorarían el modelo. Por lo tanto, hemos optado por excluir cualquier compositor con menos de 10 canciones, lo cual supone descartar tan sólo alrededor del 10 % del conjunto de datos.

En realidad, como se ha explicado, no se van a utilizar canciones completas sino canciones troceadas, lo cual resulta en la distribución de la Figura 4.3. Como se puede observar, la distribución cambia debido a la duración de las canciones. Por ejemplo, Bach tiene canciones muy cortas (la mayoría entre 100 y 200 segundos) y es por eso que al trocear, pasa de ser el segundo con más canciones en la Figura 4.2a, al cuarto en la Figura 4.3a. Además, como ya habíamos explicado, de esta forma convertimos las 330 canciones originales de MusicNet en 2143 muestras, de las cuales 1712 serán utilizadas para entrenar los modelos y 431 para realizar tests. En el caso de MAESTRO, las 1276 canciones se convierten en 11277 muestras, de las cuales 9016 serán usadas para entrenar y 2261 para testear.

Otro aspecto importante es que para mantener la generalidad del modelo, la separación de los conjuntos se realiza de forma aleatória, es decir, cada vez que se ejecuta el código tendremos el mismo número de canciones en cada conjunto (separación 80-20) pero no serán las mismas canciones que la vez anterior. Esta separación se realiza mediante la función *train_test_split* de la librería *Scikit-learn*.

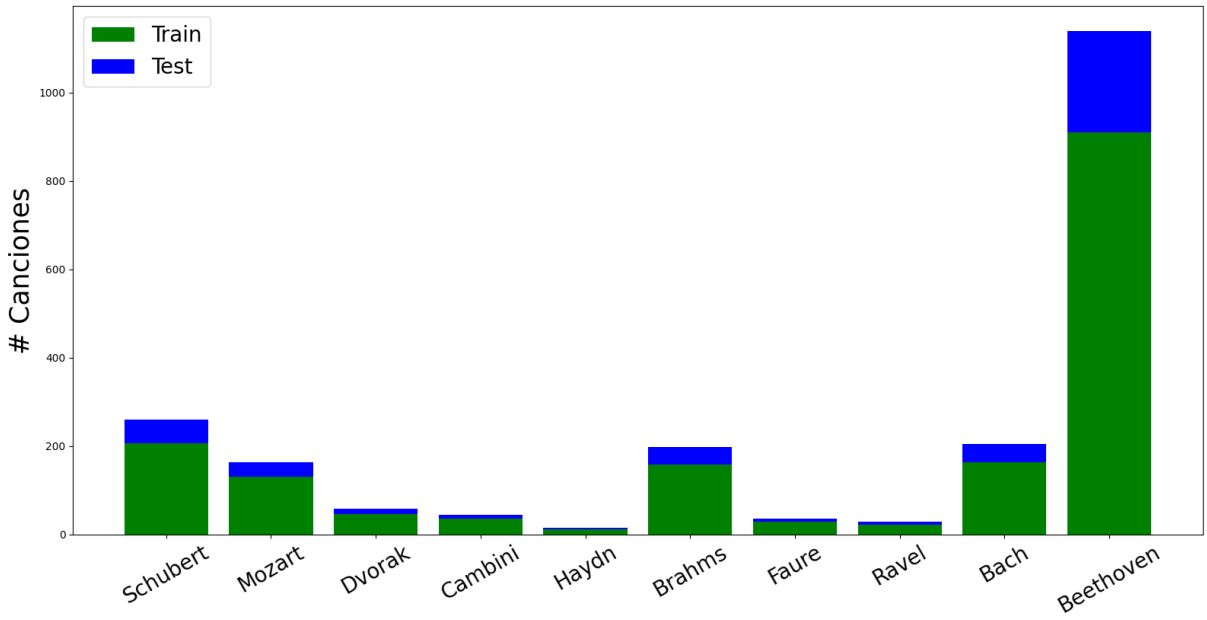


(a) MusicNet.

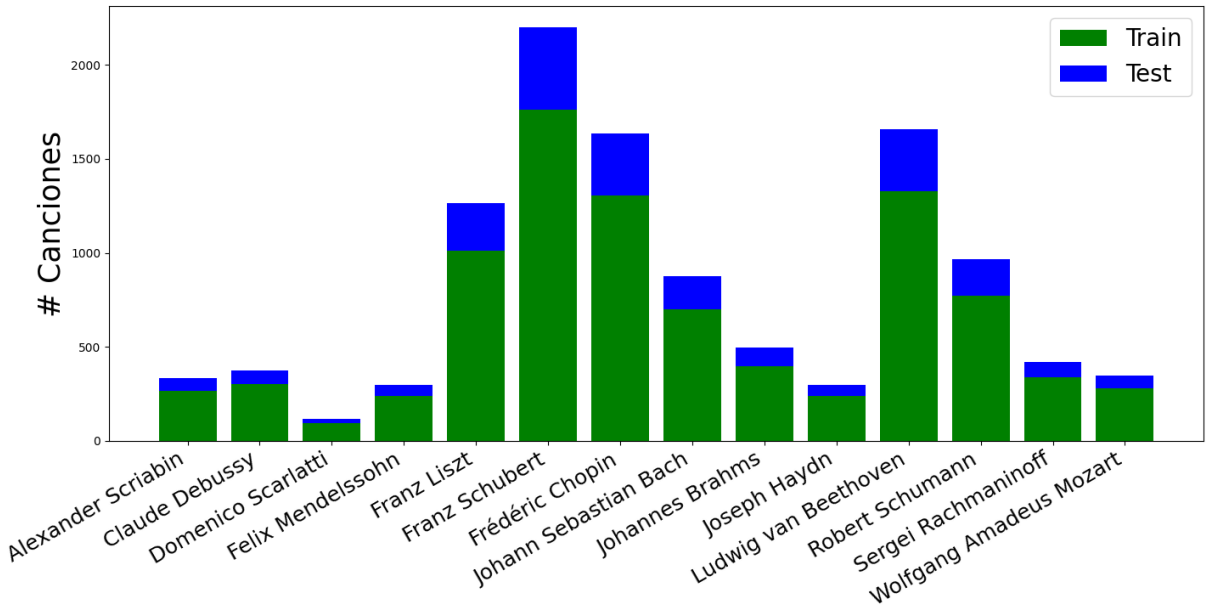


(b) MAESTRO.

Figura 4.2: Distribución de los compositores según el conjunto (azul para test y verde para entrenamiento).



(a) MusicNet.



(b) MAESTRO.

Figura 4.3: Distribución de los compositores según el conjunto (azul para test y verde para entrenamiento), utilizando el troceado de 60 segundos.

4.3. K-Nearest Neighbors

Empezaremos utilizando el modelo KNN explicado en la sección 3.2. Para ello, se usa el modelo *KNeighborsClassifier* de la librería *Scikit-learn* y las transformaciones explicadas en la sección 3.6. Como se separan las canciones al azar, para cada combinación de parámetros iteramos 10 veces y nos quedamos con el mejor resultado.

MusicNet

En las Figuras 4.4 y 4.5 se presentan los resultados obtenidos utilizando distintos pesos, métricas y número de vecinos para el conjunto de datos MusicNet. El mejor resultado ha sido:

Métrica	Peso	Vecinos	Precisión
Manhattan	Distancia	4	94.9 %

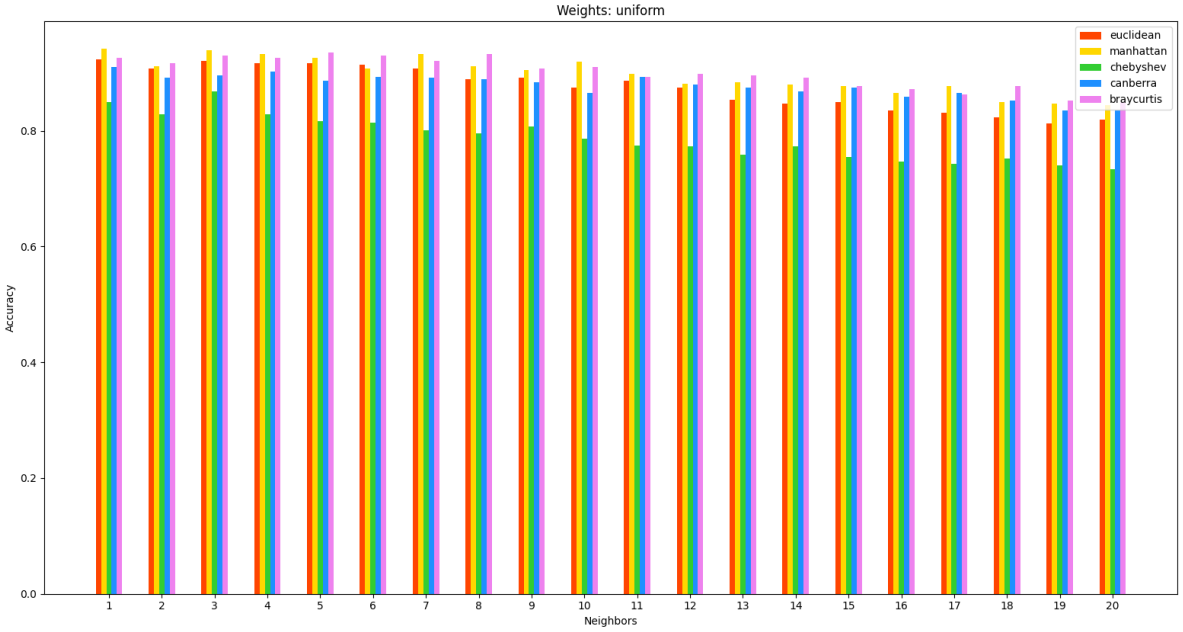


Figura 4.4: Resultados obtenidos con el modelo KNN de peso uniforme para MusicNet.

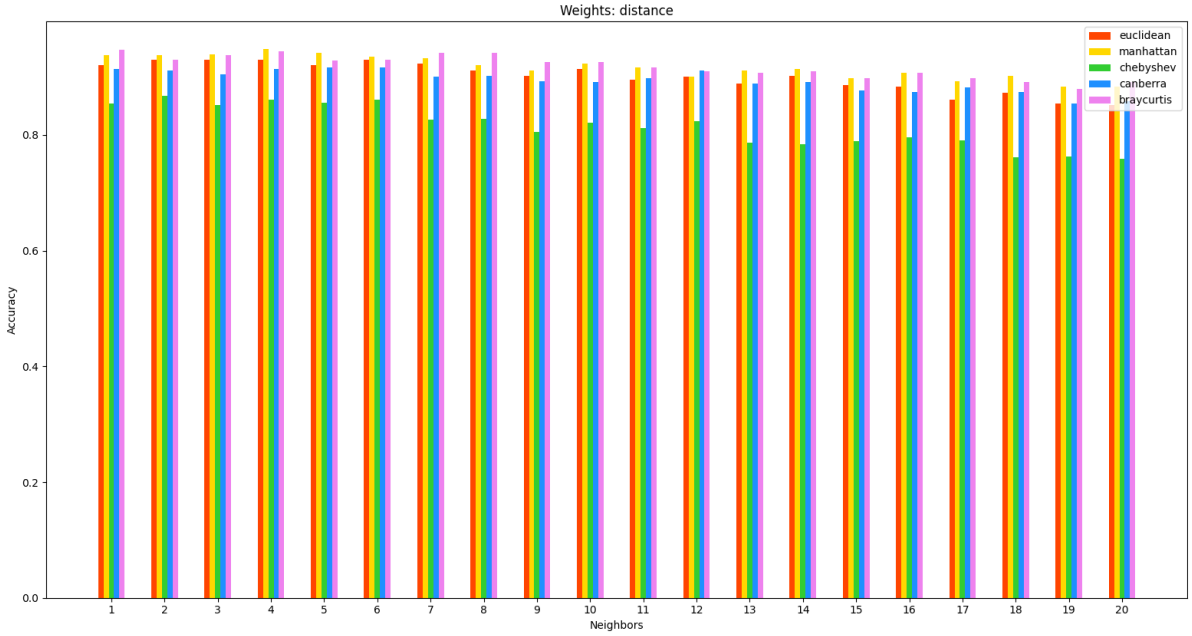


Figura 4.5: Resultados obtenidos con el modelo KNN de peso por distancia para MusicNet.

MAESTRO

En las Figuras 4.6 y 4.7 se presentan los resultados para el dataset MAESTRO. El mejor resultado obtenido ha sido:

Métrica	Peso	Vecinos	Precisión
Braycurtis	Distancia	1	75.98 %

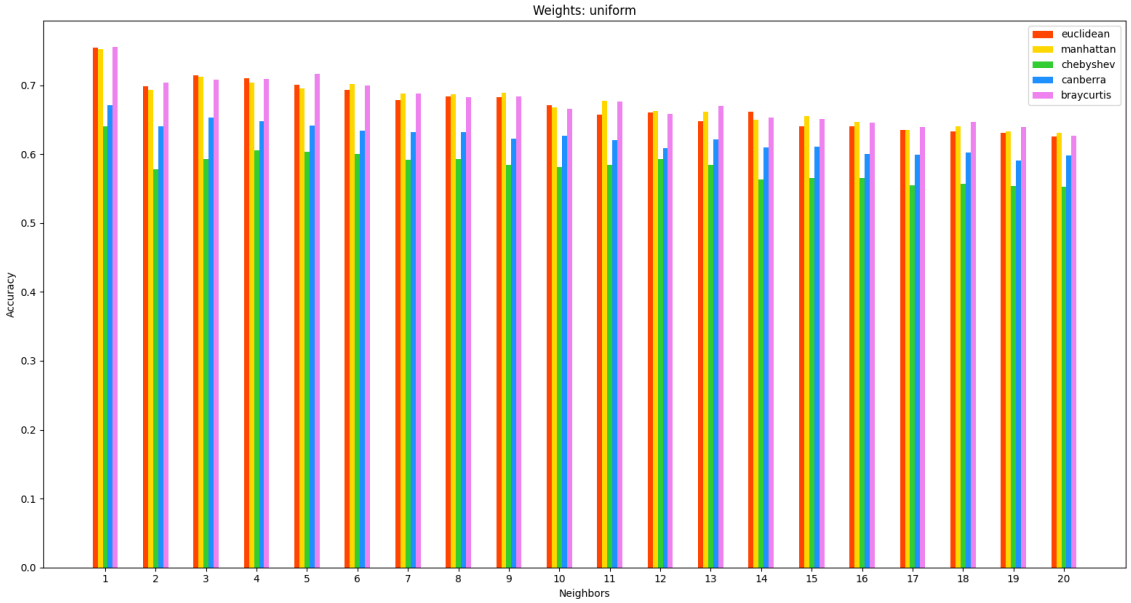


Figura 4.6: Resultados obtenidos con el modelo KNN de peso uniforme para MAESTRO.

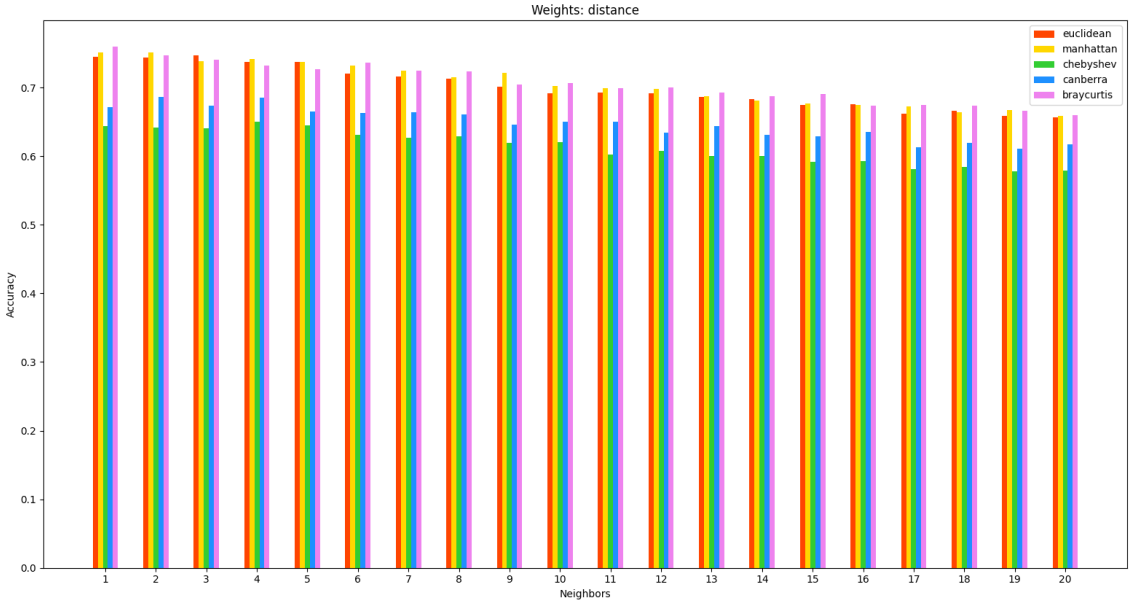


Figura 4.7: Resultados obtenidos con el modelo KNN de peso por distancia para MAESTRO.

4.4. Support Vector Classifier

En la sección 3.3 explicamos que para calcular el hiperplano se minimizan funciones usando multiplicadores de Lagrange. Un parámetro que se utiliza en la función lagrangiana es el *coste* C, que controla el margen del clasificador. A mayor C, menor es el margen y por lo tanto, más nos concentramos en los puntos cercanos a la frontera [12]. Hay que tener cuidado con este parámetro, dado que para valores muy grandes puede conllevar un *overfitting* del modelo, es decir, una pobre generalización. Este parámetro también afecta de forma distinta dependiendo del kernel. El kernel lineal es muy insensible a este parámetro, algo que podemos observar en las Figuras 4.8 y 4.9, pero otras funciones kernel son bastante más sensibles. El kernel rbf, por ejemplo, es muy sensible a este parámetro. Sin embargo, el parámetro γ se puede utilizar para contrarrestar un poco los efectos de C, efecto que viene bastante bien explicado en la documentación de *Scikit-learn* [4]. El parámetro C es usado como variable independiente (eje x), tomando valores entre 1 y 20, para obtener las Figuras 4.8 y 4.9. El resto de parámetros (como γ en el kernel *rbf*) se aplican con el valor predeterminado de la función *SVC*.

MusicNet

Las pruebas realizadas muestran que los mejores resultados se obtienen para $C \in [4, 20]$. Se ha obtenido el mejor resultado de 97.45 % con distintos valores de C, se optará por $C = 6.32$ dado que un valor cercano a 6 se ha obtenido múltiples veces:

Kernel	C	Precisión
rbf	6.32	97.45 %

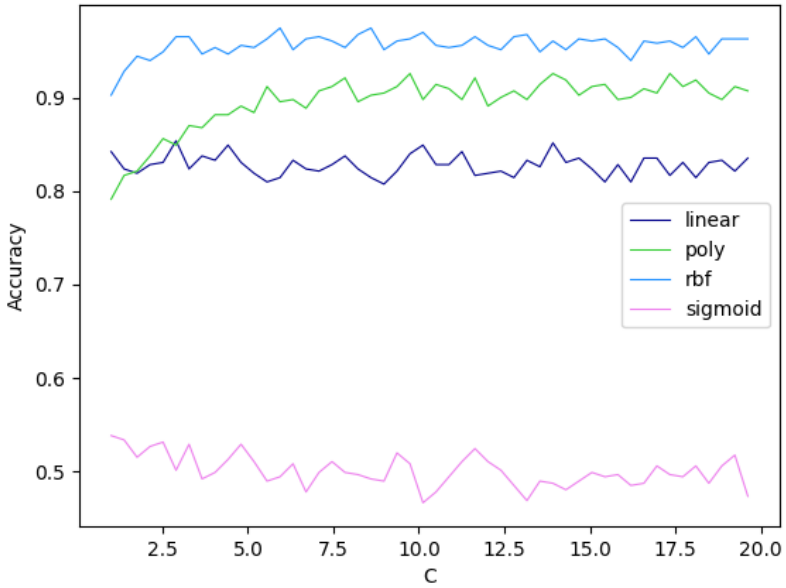


Figura 4.8: Resultados obtenidos con el modelo SVC para MusicNet.

MAESTRO

Como este dataset tiene muchas más muestras, realizar los experimentos de clasificación resulta mucho más costoso computacionalmente, por lo que se ha reducido el número de iteraciones a 5 y, dado que el kernel sigmoide no funciona nada bien para nuestra tarea, como podemos ver en la Figura 4.7, en este caso no se aplica. El mejor resultado obtenido es:

Kernel	C	Precisión
rbf	19.52	79.92 %

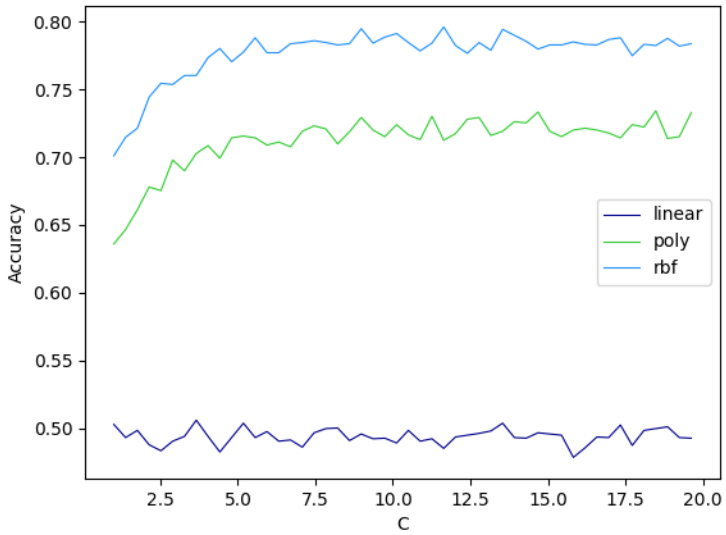


Figura 4.9: Resultados obtenidos con el modelo SVC para MAESTRO. En esta gráfica el máximo es un 79.61 % para $C = 11.64$.

4.5. Gaussian Process Classifier

Aparte de algunos parámetros para controlar el número de iteraciones y otros aspectos del algoritmo utilizado, el modelo GPC que utiliza la función *GaussianProcessClassifier* tan sólo tiene un parámetro con el cual podemos modificar el comportamiento del modelo: la función kernel. Hay varios tipos de función kernel que se pueden utilizar: *constante*, *ruido blanco*, *rbf*, *matérn*, *rational quadratic*, *exp-sine-squared* e incluso kernels obtenidos mediante la suma o producto de los anteriores y exponenciación a una potencia p [1]. Los kernels *constante* y de *ruido* se utilizan principalmente en combinación con otros kernels. Un aspecto importante es que los hiperparámetros de los kernels se optimizan durante el entrenamiento mediante una *función objetivo*, algo bastante común en distintos métodos de aprendizaje automático.

Como se puede ver en la Tabla 4.1, es posible obtener resultados decentes con distintos tipos de kernel, pero según las pruebas realizadas, el kernel *rbf* es el que mejor rendimiento ofrece en la tarea de clasificación propuesta en este trabajo.

[1] Todos estos kernels y las operaciones asociadas vienen descritos en la documentación de *Scikit-learn* [4].

MusicNet

Los mejores resultados obtenidos se resumen en la Tabla 4.1. El kernel *rbf* consigue de forma consistente alrededor de 93-94 % de precisión, mientras que otros kernels llegan a bajar hasta el 50 %.

Kernel	Precisión
rational quadratic	69.14 %
matérn	72.16 %
rbf	94.20 %

Tabla 4.1: Resultados obtenidos con modelos GPC usando distintos kernels, para el dataset MusicNet.

MAESTRO

Para este conjunto de datos se alcanza alrededor del 75 % de precisión utilizando modelos GPC con kernel *rbf*, siendo el mejor resultado un 75.90 %.

4.6. Voting Classifier

Como se ha explicado en la sección 3.5, hay dos tipos de clasificadores por votación: hard y soft. Para poder utilizar la votación soft, se hace necesario que cada modelo proporcione probabilidades de las clases. El GPC ya proporciona probabilidades de clase, y de hecho, el KNN también puede proporcionar probabilidades de forma muy sencilla, simplemente contando el número de puntos vecinos de cada clase y usando los pesos. El problema es que el modelo SVC proporciona predicciones determinísticas. Lo que generalmente se hace es lo que se conoce como *Platt scaling/calibration*. Este método se basa en aplicar una función sigmoide a la ecuación del hiperplano para obtener una distribución de probabilidad [14]:

$$p(y = 1|\mathbf{x}) = \sigma(A(\mathbf{w} \cdot \phi(\mathbf{x}) + b) + B) \tag{4.6.1}$$

donde A y B son constantes que se calculan minimizando alguna función de pérdida (como la de *entropía cruzada*, *cross-entropy* en inglés). Además, se realiza un proceso de *cross-validation* para evitar el *overfitting*.

MusicNet

Utilizando el modo de votación hard obtenemos de forma consistente un 94-95 %; el mejor resultado obtenido es de un 95.36 %. Por otro lado, utilizando el modo de votación soft obtenemos aproximadamente 96-97 %; el mejor resultado es de 97.45 % con pesos uniformes. Si se le da más importancia al modelo SVC, que es el que mejor ha funcionado, se alcanza un 98.14 % con pesos $\{w_{KNN}, w_{SVC}, w_{GPC}\} = \{0.3, 0.5, 0.2\}$.

MAESTRO

Utilizando ambos modos de votación se obtiene aproximadamente el mismo resultado: alrededor de un 79 %, usando la misma distribución de pesos utilizada para MusicNet.

4.7. Comparación y Análisis de Resultados

En principio, puede parecer que, por alguna razón, los métodos utilizados funcionan mucho mejor para el conjunto de datos MusicNet que para MAESTRO. Pero eso no es así si consideramos los números absolutos. El mejor resultado para MusicNet ha sido de un 98.14 %, que implica que se han predicho correctamente 423 de 431 muestras. Por otro lado, el mejor resultado de MAESTRO es de un 79.92 % que equivale a 1807 correctas de 2261 muestras. Si troceamos las canciones de MAESTRO utilizando una duración de 300 segundos, obtenemos 2526 canciones que separamos en 2015 para entrenamiento y 511 para test, números comparables a los de MusicNet. Si ahora realizamos el mismo análisis para cada modelo, obtenemos, de forma general, el resultado esperado: una menor precisión debido al menor número de muestras de entrenamiento. Para todos los modelos obtenemos un resultado similar o inferior, a excepción del SVC, con el cual llegamos a obtener hasta un 83.37 %, lo que equivale a 426 predicciones correctas de 511. Un resultado bastante inferior comparado con el obtenido para MusicNet.

La pregunta que nos hacemos ahora es por qué existe esta discrepancia entre los dos conjuntos. De hecho, dado que todas las canciones de MAESTRO están interpretadas utilizando un único e idéntico instrumento, es razonable esperar mejores predicciones para este conjunto que para MusicNet y sin embargo, ocurre lo contrario. Para entender por qué ocurre esto, planteamos el siguiente experimento. Hemos sido capaces de identificar (mediante el nombre, movimiento y audio) algunas canciones de Schubert y Bach que están presentes en ambos conjuntos y tocadas por un único piano. Estas canciones son las de la Tabla 4.2 con sus respectivos identificadores según el conjunto.

Compositor	Pieza	Movimiento	MusicNet	MAESTRO
Franz Schubert	Sonata en A menor	Primer	1749	22_R2_2006_01
Franz Schubert	Sonata en A menor	Segundo	1750	22_R2_2006_02
Franz Schubert	Sonata en A menor	Tercer	1751	22_R2_2006_03
Franz Schubert	Sonata en A menor	Cuarto	1752	22_R2_2006_04
Johann S. Bach	BWV 852	Primer	2225	01_R1_2008_wav–1
Johann S. Bach	BWV 852	Segundo	2224	
Johann S. Bach	BWV 865	Primer	2230	R1-D7_12
Johann S. Bach	BWV 865	Segundo	2229	

Tabla 4.2: Canciones y sus identificadores según el conjunto.

La idea es entrenar modelos utilizando todas las canciones de cada conjunto excepto los trozos correspondientes a las canciones de la Tabla 4.2 y realizar una "predicción cruzada", es decir, ver si el modelo entrenado con MusicNet consigue predecir las muestras de MAESTRO y viceversa. Si hacemos esto únicamente para las muestras de Schubert, obtenemos unos resultados impresionantes (Tabla 4.3). Los modelos entrenados con las muestras de MAESTRO predicen extremadamente bien, incluso predice las muestras de MusicNet mejor que el propio MusicNet. Este es el resultado que esperábamos, que un modelo entrenado con muestras tocadas por un único piano clasifique mucho mejor que uno entrenado con varios instrumentos. Sin embargo, esto se puede deber a que las muestras de Schubert suponen únicamente el 12 % de MusicNet (259/2143) mientras que en MAESTRO llega casi al 20 % (2202/11277). Por lo tanto, MAESTRO podría estar prediciendo mejor Schubert simplemente porque está más entrenado para predecir Schubert.

Modelo	Conjunto de Entrenamiento	Conjunto de Test	Precisión
KNN	MusicNet	MusicNet	46.15 %
	MusicNet	MAESTRO	35.14 %
	MAESTRO	MAESTRO	97.30 %
	MAESTRO	MusicNet	69.23 %
SVC	MusicNet	MusicNet	56.41 %
	MusicNet	MAESTRO	64.86 %
	MAESTRO	MAESTRO	89.19 %
	MAESTRO	MusicNet	53.85 %
GPC	MusicNet	MusicNet	41.03 %
	MusicNet	MAESTRO	24.32 %
	MAESTRO	MAESTRO	100.00 %
	MAESTRO	MusicNet	74.36 %
VC	MusicNet	MusicNet	58.97 %
	MusicNet	MAESTRO	62.16 %
	MAESTRO	MAESTRO	91.89 %
	MAESTRO	MusicNet	61.54 %

Tabla 4.3: Resultados obtenidos según el conjunto y modelo para las muestras de Schubert. Los modelos usados utilizan los mejores parámetros que encontramos en sus respectivas secciones para cada conjunto.

Pero este no es exactamente el caso. En la Tabla 4.4 podemos ver que MAESTRO sigue funcionando bastante bien al predecir muestras de Bach que solo componen el 7.75 % (874/11277) del conjunto de entrenamiento, cifra más comparable al 9.57 % de MusicNet (205/2143). Por lo tanto, todo indica que los peores resultados obtenidos para MAESTRO se deben simplemente al volumen y distribución de los datos. MAESTRO no sólo contiene más compositores sino que también presenta una distribución mucho más desequilibrada (Figura 4.3b). MusicNet tiene una distribución bastante más uniforme si exceptuamos a Beethoven (Figura 4.3a). Si sólo consideramos los 4 compositores con mayor número de muestras de MAESTRO, lo cual supone un 60 % del conjunto, se puede mejorar la precisión en hasta un 10 %.

Como podemos observar en las Tablas 4.3 y 4.4, los modelos entrenados con un conjunto predicen peor las canciones del otro conjunto, algo completamente razonable dado que no sólo se utilizan pianos distintos sino que, y más importante, las interpretaciones son muy distintas. Esto se hace evidente al escuchar las canciones. Si nos fijamos en el tempo, por ejemplo, las canciones de MAESTRO suelen ser más rápidas. De hecho, si quitamos el tempo de los rasgos que utilizamos, la precisión de las predicciones en el propio conjunto disminuye pero la del otro conjunto aumenta en hasta un 10 %. MAESTRO consigue una buena predicción para Schubert no sólo por que está más entrenedado, sino también porque las interpretaciones de MusicNet son similares a las de MAESTRO, mientras que con Bach ocurre lo contrario.

Otro resultado importante, que no se ve reflejado en las Tablas 4.3 y 4.4, es que todas las predicciones de MusicNet son Schubert, Bach o Beethoven. Cuando obtenemos 58.97 % de acierto para Schubert utilizando el modelo VC, significa que 23 muestras las clasificamos correctamente como Schubert mientras que las 16 restantes las clasificamos

Modelo	Conjunto de Entrenamiento	Conjunto de Test	Precisión
KNN	MusicNet	MusicNet	50.00 %
	MusicNet	MAESTRO	33.33 %
	MAESTRO	MAESTRO	91.67 %
	MAESTRO	MusicNet	33.33 %
SVC	MusicNet	MusicNet	83.33 %
	MusicNet	MAESTRO	16.67 %
	MAESTRO	MAESTRO	83.33 %
	MAESTRO	MusicNet	58.33 %
GPC	MusicNet	MusicNet	41.67 %
	MusicNet	MAESTRO	16.67 %
	MAESTRO	MAESTRO	75.00 %
	MAESTRO	MusicNet	8.33 %
VC	MusicNet	MusicNet	83.33 %
	MusicNet	MAESTRO	33.33 %
	MAESTRO	MAESTRO	91.67 %
	MAESTRO	MusicNet	58.33 %

Tabla 4.4: Resultados obtenidos según el conjunto y modelo para las muestras de Bach. Los modelos usados utilizan los mejores parámetros que encontramos en sus respectivas secciones para cada conjunto.

como Beethoven. Para otros modelos se predice Bach ocasionalmente, pero los fallos son, en su mayoría, predicciones de Beethoven. Esto es algo bastante sencillo de explicar si consideramos la distribución de muestras tocadas con un único piano. Como podemos ver en la Figura 4.10, de los 10 compositores de MusicNet, sólo Schubert, Bach y Beethoven tienen canciones interpretadas con un único piano, lo cual explica que estas sean las únicas predicciones posibles de MusicNet. Además, las muestras de Beethoven suponen más del doble que las de los otros dos juntos, lo cual explica que la mayor parte de predicciones erróneas sean de Beethoven.

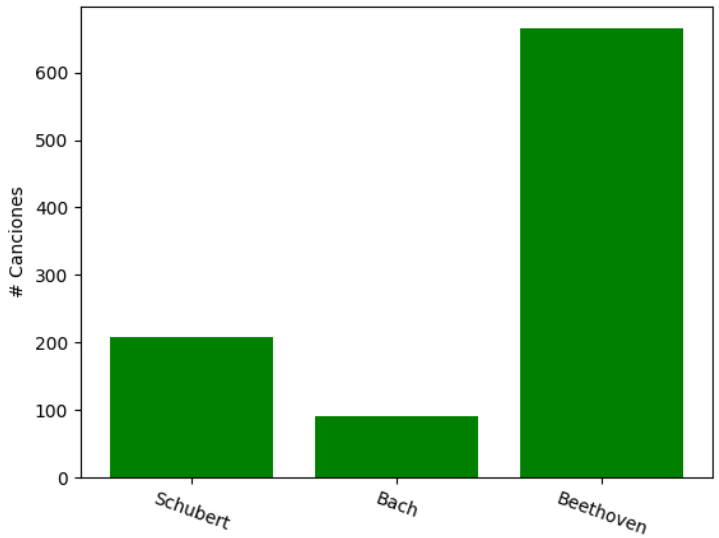


Figura 4.10: Distribución de muestras obtenidas de canciones de MusicNet tocadas con un único piano.

4.8. Comprobación del Modelo

Para comprobar que el mejor modelo de MusicNet obtenido, el modelo VC (sección 4.6), obtiene buenos resultados con otros conjuntos similares, se puede estudiar su rendimiento al aplicarlo a distintos troceados de MusicNet (trocear las canciones con distintas duraciones). En la Tabla 4.5 podemos observar los resultados obtenidos. Como se puede ver, se sigue obteniendo unos resultados muy buenos para un gran número de muestras de test. De hecho, con el troceado de 10 segundos disponemos de más muestras que el troceado del conjunto MAESTRO que se ha utilizado en secciones anteriores y seguimos obteniendo un rendimiento muy superior.

Duración del Troceado	Muestras de Entrenamiento / Test	Precisión
60 s	1712 / 431	98.14 %
50 s	2043 / 515	96.70 %
40 s	2529 / 636	95.91 %
30 s	3351 / 843	95.85 %
25 s	4005 / 1005	95.22 %
20 s	4989 / 1252	94.73 %
15 s	6627 / 1662	94.71 %
10 s	9907 / 2482	93.76 %

Tabla 4.5: Resultados obtenidos para distintas duraciones del troceado de las canciones de MusicNet utilizando el modelo VC de la sección 4.6.

Una tendencia que podemos observar en la Tabla 4.5 es que la precisión disminuye de forma muy lineal. En la Figura 4.11 se puede observar un ajuste lineal realizado a los resultados de la Tabla 4.5. Se obtiene un coeficiente de correlación $R = 0.9745$, lo que indica que hay una relación subyacente entre la duración del troceado y la precisión del modelo. La razón de la existencia de dicha relación es un tema muy complejo debido al gran número de rasgos y su variabilidad con la duración del troceado. De forma cualitativa, esto podría deberse a que la menor duración de los trozos impide el reconocimento de patrones musicales característicos de cada compositor y tan sólo se consigue identificar patrones más comunes formados por unas pocas notas o acordes.

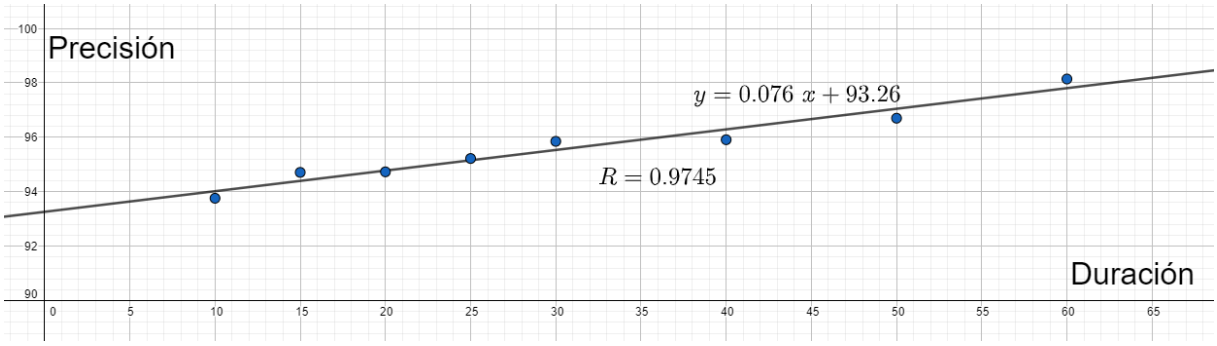


Figura 4.11: Relación precisión/duración de la Tabla 4.5.

Capítulo 5

Conclusiones

El objetivo de este trabajo es el de obtener un modelo sólido de machine learning con el cual clasificar canciones de música clásica según el compositor. Se ha analizado el uso de los modelos KNN, SVC, GPC y VC para dos conjuntos de datos distintos obteniendo unos resultados bastante positivos. Además, se ha realizado una comparación cualitativa de los modelos obtenidos mediante cada conjunto de datos.

El mejor rendimiento de los modelos para MusicNet lleva a la conclusión de que un conjunto de entrenamiento formado por canciones interpretadas por una configuración variada de instrumentos se adapta mejor a la tarea de clasificación general. La razón por la que se produce este resultado es un tema complicado debido al gran número de variables. De forma cualitativa, se podría decir que la presencia de canciones interpretadas con múltiples y distintos instrumentos permite la identificación de sesgos frecuenciales más relacionados con patrones musicales de cada compositor que con el timbre del instrumento. De forma cuantitativa, habría que estudiar los efectos de cada uno de los 36 rasgos que se han utilizado y comparar los rasgos obtenidos utilizando ambos conjuntos. Como se ha mencionado, la eliminación de algunos rasgos puede suponer una mejora en determinados casos.

Por otra parte, se ha comprobado que un conjunto de entrenamiento interpretado por un único instrumento puede ser útil y bastante eficaz en clasificaciones de canciones interpretadas con dicho instrumento. En cualquier caso, los modelos son bastante susceptibles a fallos debido a distintas interpretaciones. Es decir, la misma canción tocada por distintos músicos suele presentar diferencias importantes de tempo, duración de pausas, *staccato*, *legato*, *glissando*, etc. que conllevan el fallo en la clasificación.

La conclusión es que los algoritmos de machine learning son ideales para la clasificación musical siempre que se disponga de un conjunto de entrenamiento adecuado. Este conjunto debería de tener un gran volumen de canciones interpretadas utilizando una selección variada de instrumentos comunes en la música clásica (violín, viola, piano, flauta, ...) y canciones repetidas interpretadas por distintos músicos.

Bibliografía

[1] Magenta. "MAESTRO Dataset." (mayo de 2022), dirección: https://magenta.tensorflow.org/datasets/maestro#download.

[2] S. GUPTA. "MusicNet Dataset." (mayo de 2022), dirección: https://www.kaggle.com/datasets/imsparsh/musicnet-dataset.

[3] B. McFee, A. Metsai, M. McVicar et al., *librosa*, ver. 0.9.1, feb. de 2022. DOI: 10.5281/zenodo.6097378. dirección: https://doi.org/10.5281/zenodo.6097378 (visitado 05-2022).

[4] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011. dirección: https://scikit-learn.org/stable/index.html (visitado 05-2022).

[5] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer International Publishing, 2015, ISBN: 9783319219455.

[6] C. Schörkhuber y A. Klapuri, "Constant-Q transform toolbox for music processing," *Proc. 7th Sound and Music Computing Conf.*, págs. 322-329, ene. de 2010.

[7] E. Schubert, J. Wolfe y A. Tarnopolsky, "Spectral centroid and timbre in complex, multiple instrumental textures," *8th International Conference on Music Perception and Cognition*, págs. 654-657, ago. de 2004.

[8] T. Ganchev, N. Fakotakis y K. George, "Comparative evaluation of various MFCC implementations on the speaker verification task," *Proceedings of the SPECOM*, vol. 1, págs. 191-194, ene. de 2005.

[9] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, n.º 3, págs. 90-95, 2007. DOI: 10.1109/MCSE.2007.55. dirección: https://matplotlib.org/stable/index.html (visitado 05-2022).

[10] M. L. Waskom, "Seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, n.º 60, pág. 3021, 2021. DOI: 10.21105/joss.03021. dirección: https://doi.org/10.21105/joss.03021 (visitado 05-2022).

[11] C. Cortes y V. N. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, págs. 273-297, 2004.

[12] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, ép. Springer Series in Statistics. Springer New York, 2009, ISBN: 9780387848587.

[13] C. Rasmussen y C. Williams, *Gaussian Processes for Machine Learning*, ép. Adaptative computation and machine learning series. University Press Group Limited, 2006, ISBN: 9780262182539.

[14] C. Bishop, *Pattern Recognition and Machine Learning*, ép. Information Science and Statistics. Springer, 2006, ISBN: 9780387310732.

[15] The Pandas development team, *Pandas*, ver. latest, feb. de 2020. DOI: 10.5281/zenodo.3509134. dirección: https://doi.org/10.5281/zenodo.3509134 (visitado 05-2022).