Trabajo Fin de Grado
Ingeniería de Tecnologías de Telecomunicación

Reconocimiento automático de instrumentos mediante aprendizaje máquina

Autor: Aurora Salgado Díaz del Río

Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Ingeniería de Tecnologías de Telecomunicación

Reconocimiento automático de instrumentos mediante aprendizaje máquina

Autor:
Aurora Salgado Díaz del Río

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado: Reconocimiento automático de instrumentos mediante aprendizaje máquina

Autor: Aurora Salgado Díaz del Río
Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, quiero agradecer todo el apoyo que he tenido por parte de mi familia, en especial el de mi madre y el de mi padre, quienes me han enseñado las dos lecciones más grandes que he aprendido: ”el esfuerzo determina tu suerte” y ”no importa las veces que te caigas, siempre te vas a levantar”.

A mi hermano por enseñarme qué es la verdadera paciencia.

A mis amigas Aurora, Coral, Xiao y Laura. Ellas sí que han llegado a entender que es una carrera de ingeniería sin ni siquiera vivirla.

A mis amigos de la carrera, compañeros de sufrimiento, en especial a los que me han acompañado en todo el camino, Pablo, Lucas, Enrique y Ale.

A mis amigos de la rama, Pepe, Carmelo y Gata por ayudarme a confiar en mí misma y conseguir que no pierda la fe en lo que creo que es lo correcto.

A mis niñas de teleco, Carmen, Alexa, Asun, jamás estaría aquí si no las hubiera conocido.

A Fernando y a Enrique por su paciencia, por estar en todos los momentos, buenos o malos, ellos siempre han estado cuando les necesitaba.

A Pablo Monleón, mi mejor amigo desde siempre. Jamás podré agradecerte todo lo vivido.

A todos mis profesores, en especial a Francisco, por la paciencia que ha tenido conmigo, por aguantar todas las dudas infinitamente largas, los e-mails interminables y mis continuas frustraciones.

Aurora Salgado Díaz del Río
Sevilla, 2019

Resumen

En este proyecto se pretende abordar el análisis de los algoritmos de Aprendizaje Máquina con el objetivo de clasificar automáticamente sonidos de distintos instrumentos y a su vez razonar cuales son las características más significativas para poder realizar un reconocimiento óptimo de cada uno. En este caso particular, los instrumentos a analizar son: el violín, el piano eléctrico, el saxofón, el clarinete, la guitarra y el oboe. Para ello se han extraído sesenta características de cada uno de los audios a clasificar y posteriormente se han utilizado técnicas de aprendizaje automático para realizar el reconocimiento de los instrumentos. En concreto, se ha utilizado los algoritmos PCA, LDA, KNN, Random Forest y redes neuronales multicapa. Por último se verá una tabla de los resultados óptimos con sus respectivos algoritmos y parámetros.

Abstract

The aim of this project is to approach the analysis of Machine Learning algorithms in order to classify automatically sounds from different instruments and at the same time to justify which are the most significant features in order to perform an optimal recognition of each of the instruments to be classified. In this particular case, the instruments to be analyzed are: the violin, the electric piano, the saxophone, the clarinet, the guitar and the oboe. Futhermore, sixty features of each of the audios to be classified have been extracted and later automatic learning techniques have been used to perform the recognition of the instruments. In particular, the algorithms are PCA, LDA, KNN, Random Forest and multilayer neural networks. layer neural networks. Finally there is a table of the optimal results with their respective algorithms and parameters.

Índice Abreviado

Índice

1 Introducción

El entrenamiento musical es un instrumento más potente que cualquier otro porque el ritmo y la armonía encuentran su camino en lo más profundo del alma

Platón

El reconocimiento automático de instrumentos musicales forma parte de la investigación del audio desde hace muchos años. El gran avance en los sistemas inteligentes y en la potencia computacional de los ordenadores actuales ha conseguido que los resultados sean mucho más precisos y coherentes que en anteriores investigaciones, no obstante, esta clasificación sigue siendo objeto de estudio y está en continua evolución. La clasificación de instrumentos mediante técnicas de aprendizaje automático pretende imitar el oído humano.
Existe una amplia variedad de algoritmos de Machine Learning que son capaces de analizar la correlación entre las diferentes características espectrales que diferencian dichos instrumentos. Las técnicas habituales son: los árboles de decisión, la clasificación K vecinos más cercanos (KNN), el clasificador Bayesiano ingenuo (Naives Bayes), las máquinas de soporte vectorial (SVM), la clasificación de agregados independientes (Random Forest) y las redes neuronales artificiales (ANN). En este proyecto las herramientas utilizadas para llevar a cabo esta clasificación han sido los algoritmos de redes neuronales, KNN y Random Forest. Además se han utilizado algoritmos para extraer las características más significativas de cada audio como el *análisis de los componentes principales* (PCA) o el *análisis discriminante lineal* (LDA).

1.1 Objetivos

La finalidad de este proyecto es la comparación de algoritmos de aprendizaje automático para encontrar con cuál se obtiene un funcionamiento óptimo. Se pretende hacer un reconocimiento de seis instrumentos. Los instrumentos a analizar son los siguientes:

- Guitarra
- Violín
- Piano eléctrico
- Saxofón
- Oboe
- Clarinete

En la realización de esta clasificación es necesaria la creación de una extensa base de datos compuesta por una gran cantidad audios de libre disposición. Para ello se ha usado *Freesound* [6], una página web en la que se pueden descargar una gran variedad de audios tanto de instrumentos de todas las familias, como sonidos urbanos, música de diferentes estilos musicales etc. Así mismo la extracción de las características principales que hacen a cada instrumento único, ha podido ser realizada gracias *Essentia* [5], una librería de Python

desarrollada por el ”Music Technology Group” también conocido como MTG, perteneciente a la Universidad Pompeu Fabra. Esta herramienta se desarrollará con más detalle en el capítulo cuatro (Explicación concreta de las herramientas, datos y algoritmos usados). Se ha de tener en cuenta que con la librería Essentia se extraen todas las características de un audio, por lo tanto se debe hacer un análisis que encuentre cuales de estas son las más descriptivas para cada instrumento, es decir, las que marcan la diferencia entre cada uno de ellos. Una vez se realiza la extracción de características más significativas de estos audios, se deberá realizar un fichero de extensión ”.csv” en el cual se incorporaran las características anteriormente expuestas con el fin de que sea implementable en los algoritmos de aprendizaje automático.
El objetivo final del proyecto es evaluar los resultados a través de una serie de modificaciones. Se implementarán distintos algoritmos de inteligencia artificial, y además se modificarán sus parámetros con el fin de identificar cuales son las debilidades de estos algoritmos y cuales serían las posibles mejoras.

1.2 Organización de la memoria del proyecto

1. **Introducción:** Breve resumen de la finalidad de este trabajo, exponiendo así mismo los objetivos principales a realizar y la estructura de la memoria que se va a llevar a cabo.
2. **Inteligencia Artificial:** En este capítulo se definirán los conceptos de Inteligencia Artificial (IA) y de Machine Learning (ML), también se hará una introducción a los algoritmos actuales usados en los sistemas inteligentes más modernos.
3. **Aplicaciones de la inteligencia artificial en el análisis de datos musicales:** Estado del arte actual en relación a las nuevas tendencias de análisis de datos musicales.
4. **Explicación concreta de las herramientas, datos y algoritmos utilizados:** En este capítulo se matizaran los set de datos usados, así como los entornos en los que se ha trabajado, las librerías de Python y por último se hará una definición más concreta del funcionamiento de los algoritmos utilizados para la realización del proyecto.
5. **Resultados:** Se realizará una batería de pruebas con los diferentes sets de datos, modificaciones tanto de algoritmos utilizados como parámetros dentro de los mismos. Con el único fin de fijar los parámetros más óptimos para nuestro sistema.
6. **Conclusiones y líneas futuras de investigación:** En este capítulo, se recogerán las partes principales del proyecto así como el trabajo futuro que podría ser realizado a partir de esta investigación.
7. **Anexo:** Por último, se expondrán los códigos de algunas de las pruebas realizadas.

2 Inteligencia Artificial

Hasta la fecha, no se ha diseñado un ordenador que sea consciente de lo que está haciendo; pero, la mayor parte del tiempo, nosotros tampoco lo somos

Marvin Minsky

Desde hace más de cincuenta años, el análisis de grandes conjuntos de datos ha ampliado sus horizontes en todos los dominios de estudio. Se ha desarrollado visiblemente en el campo del comercio, de los transportes, de la distribución y así mismo en la música.
El término Big Data actual tiene varias definiciones, todas ellas están adjuntas al significado original de la información y del procesamiento de datos. Una de las definiciones más comunes se relaciona con el principio de las 5V: volumen, velocidad, variedad, veracidad y valor de los datos explotados. La caída en los precios de almacenamiento y el aumento en la capacidad de cómputo son el origen del gran volumen y la alta velocidad del procesamiento de datos. La variedad de datos (imágenes, textos, bases de datos, objetos conectados, etc.) se debe principalmente a la creciente digitalización de los medios de información. La veracidad, para la cual fluye el valor del trabajo, es un problema central para cualquier proyecto de análisis de datos automatizado. De hecho, un algoritmo es aún más poderoso ya que los datos son numerosos, precisos y están bien adaptados a la pregunta que se va a resolver [10]. Pero para caracterizar aún más el aspecto innovador que trae Big Data, debemos hablar, por supuesto, de un subcampo de inteligencia artificial: el aprendizaje automático. Esto último hace posible construir algoritmos capaces de acumular conocimiento e inteligencia de la experiencia sin ser guiados humanamente durante su aprendizaje, ni programados explícitamente para manejar esta o aquella tarea en particular.

2.1 ¿Qué es la IA?

El término de ”Inteligencia Artificial” es relativamente actual, sin embargo, no significa que esta idea no haya existido desde hace bastantes años atrás [44]. El primer intento definir la IA fue realizado por Alan Turing, un científico ingles reconocido por inventar la máquina de Turing, la cual sigue siendo utilizada para formalizar los conceptos de modelos computacionales. En el año 1950 publicó un artículo donde argumentaba que una máquina podría considerarse inteligente si actúa como un humano [42]. Turing desarrolló una prueba para demostrar si una máquina puede considerarse que ”piensa”. Esta consiste en:

1. Se necesitan dos personas y una computadora; una persona será el interrogador mientras que la otra junto a la máquina serán los elementos a identificar.
2. Tanto el interrogador como los elementos a identificar estarán en un cuarto distinto.
3. Se realiza una comunicación escrita entre ambos elementos sin tener contacto visual.
4. Se realizan ciertas preguntas por parte del interrogador que deberán ser respondidas algunas veces por la máquina y otras tantas por la otra persona. Si el interrogador no es capaz de reconocer cuál es la computadora y quién es la persona, entonces se puede considerar que la máquina es inteligente.

Sin embargo, no fue hasta 1957 cuando un grupo de científicos que estudiaron el funcionamiento del cerebro humano como modelo natural integrando las computadoras y la cibernética le dieron una definición. La Inteligencia Artificial se definió entonces como el estudio de las facultades mentales a través de modelos computacionales.
Uno de estos científicos, Marvin Minsky, la expuso como ”el estudio de cómo programar computadoras que posean la facultad de hacer aquello que la mente humana puede realizar.”. Otra definición muy extendida es la de Hayes, el cual consideró que la propia IA es la implementación de razonamientos inteligentes mediante técnicas propias de la computación.
Un programa considerado inteligente será capaz de aprender de su experiencia mediante la generalización y abstracción, simulando el pensamiento humano, sobretodo cuando está en un entorno de incertidumbre o con información incompleta. De esta forma, modela y maneja una gran variedad de sistemas complejos construyendo una herramienta óptima para resolver problemas de gran complejidad. El programa realiza una decisión teniendo en cuenta tanto un razonamiento aproximado como el reconocimiento de patrones [44].
El término inteligencia artificial engloba varios campos tales como el *Deep Learning*, *Representation Learning* y *Machine Learning* como se puede observar en la siguiente figura:

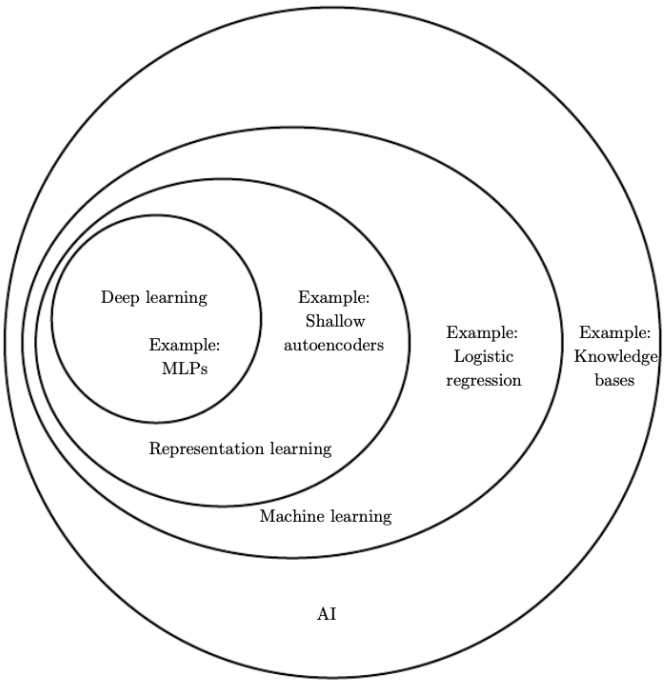

Figura 2.1 Diagrama de Venn [35].

Esta figura muestra cómo el aprendizaje profundo es un tipo de aprendizaje de representación, que a su vez es un tipo de aprendizaje automático, que son utilizados en la IA. Cada sección del diagrama de Venn incluye un ejemplo de una tecnología de inteligencia artificial [35].

2.2 Machine Learning

Machine learning (ML) es un subcampo de la Inteligencia Artificial. Se puede definir como una forma de estadística aplicada con un mayor énfasis en el uso de las computadoras para hacer estimaciones de funciones estadísticamente complicadas y obtener intervalos de confianza en torno a ellas. La mayoría de los algoritmos de aprendizaje automático tienen parámetros definidos como *hiperparámetros*, que deben determinarse fuera del propio algoritmo de aprendizaje. Machine Learning nos permite afrontar tareas que son muy complicadas para ser resueltas por el hombre. Desde un punto científico y filosófico el aprendizaje automático es muy interesante, ya que para desarrollarlo primero se debe tener una profunda idea de cómo funciona la inteligencia humana. Las tareas resueltas por ML están descritas por cómo una máquina debe

procesar un set de características también denominado *ejemplo*. Un ejemplo normalmente está representado como un vector $x \in \mathbb{R}^n$donde cada entrada del vector x_i son características. Por ejemplo, las características de un instrumento pueden ser el tono, el ritmo o la intensidad. Para evaluar cómo de óptimo es un algoritmo de ML, se debe diseñar una medida cuantitativa de su rendimiento. Se utiliza entonces una medida de precisión del modelo, es decir, cuánta probabilidad de acierto es obtenida. La precisión está relacionada con el ratio de error también denominado "pérdidas".
Por lo general, el interés principal es saber cómo de bien funciona el algoritmo de aprendizaje automático en datos que no se hayan visto anteriormente, ya que esto determina cómo va a funcionar cuando se implemente en el mundo real. Por lo tanto, evaluamos estas medidas de rendimiento utilizando un conjunto de datos llamados "datos de test-validación" que están separados de los datos utilizados para entrenar el sistema de aprendizaje automático y a partir de estos, podemos determinar la precisión de nuestro sistema.
La mayoría los algoritmos de ML se pueden dividir en dos categorías: aprendizaje supervisado y aprendizaje no supervisado. Esta clasificación depende del tipo de experiencia que se les permite tener durante el proceso de aprendizaje. La experiencia puede ser entendida en este proyecto como el set de datos que se van a analizar mientras que, un set de datos es la colección de muchas características. [35].
A continuación, se realizará una separación entre los dos tipos de algoritmos que existen, y se explicarán los dos algoritmos utilizados para la realización de este proyecto: PCA (Principal Components Analysis o Análisis de los componentes principales), LDA (Lineal discriminant analysis o análisis discriminante lineal), Random Forest, KNN (K vecinos más cercanos) y ANN (Artificial Neural Networks o redes neuronales artificiales).

2.3 Tipos de algoritmos

Los algoritmos utilizados en el aprendizaje automático se pueden clasificar en dos conjuntos: aprendizaje no supervisado y aprendizaje supervisado. A continuación se realizará una descripción de cada uno de ellos.

2.3.1 Algoritmos de aprendizaje no supervisado

La idea principal de estos algoritmos se basa en la representación de los datos de entrada de tal manera que se refleje la estructura estadística que los define, es decir, trata de buscar características o patrones que son significativos en los datos de entrada [45] . Se utiliza mayoritariamente para la compresión de datos, esto es realizado a través de agrupaciones en clústeres, que consiste en dividir el set de datos en grupos más comprimidos de características similares [35].
Un ejemplo de algoritmo no supervisado es PCA, el cual ha sido utilizado para comprimir los datos utilizados en el reconocimiento de instrumentos, obteniendo mejores resultados en precisión.

2.3.2 Algoritmos de aprendizaje supervisado

Este tipo de aprendizaje es utilizado con el fin de modelar la correspondencia entre un conjunto de N ejemplos o patrones de entrada $\left\{\mathrm{x}^1, \mathrm{x}^2, \ldots, \mathrm{x}^N\right\}$ y las salidas deseadas $\left\{\mathrm{t}^1, \mathrm{t}^2, \ldots, \mathrm{t}^N\right\}$, minimizando para ello función de error que mide la diferencia existente entre estas salidas y las obtenidas por la red $\left\{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\right\}$ [45].
La mayor diferencia que tiene con respecto al aprendizaje no supervisado es que dentro del set de datos tiene asociado un objetivo o etiqueta [35]. En nuestro caso, estas etiquetas serán los instrumentos que vamos clasificar (piano eléctrico, violín, oboe, clarinete, saxofón y guitarra).

2.4 Análisis de los componentes principales

El algoritmo de PCA deriva del algebra lineal. Se tiene un conjunto de m puntos $\left\{x^{(1)}, \ldots, x^{(m)}\right\}$ en $\mathbb{R}^n$ y se quiere hacer una compresión de ellos.
Una forma de codificar estas características es representarla en una versión de menor dimensión. Para cada punto $x^{(i)} \in \mathbb{R}^n$ se intentará encontrar un vector codificado $c^{(i)} \in \mathbb{R}^l$. Si l es menor que n, guardar la memoria de los puntos codificados significa un menor uso de memoria que utilizando los datos originales. Se busca una función de codificación que produce el código con una entrada, $f(x) = c$, y una función de decodificación que reconstruye una entrada dando su código $x \approx g(f(x))$. Este algoritmo debe además eliminar las relaciones no lineales de un set de características. A continuación se realizará un estudio de cómo funciona algebraicamente

PCA. Considerando que nuestro set de datos tiene dimensiones $m \times n$ dentro de una matriz denominada $\mathbf{X}$. Asumimos que los datos tienen de media cero, $\mathbb{E}[\mathbf{x}] = \mathbf{0}$.

La matriz de covarianza no sesgada asociada a $\mathbf{X}$ está dada por:

$$\text{Var}[\mathbf{x}] = \frac{1}{m-1}\mathbf{X}^\top\mathbf{X} \tag{2.1}$$

Los principales componentes de una matriz X están dados por los autovectores de $\mathbf{X}^\top\mathbf{X}$.. Entonces se cumple que:

$$\mathbf{X}^\top\mathbf{X} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top \tag{2.2}$$

PCA encuentra una representación (por transformación lineal) $\mathbf{z} = \mathbf{W}^\top\mathbf{x}$, donde Var $[\mathbf{z}]$ es la diagonal.

Los componentes principales se pueden obtener por descomposición de valores singulares (SVD). Específicamente, son los valores singulares de $\mathbf{X}$. Para ver esto, supongamos $\mathbf{W}$ ser un valor singular correcto en descomposición de $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top$. Se puede entonces reconstruir la ecuación del autovector original con $\mathbf{W}$ como la correspondiente base asociada al autovector:

$$\mathbf{X}^\top\mathbf{X} = \left(\mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top\right)^\top \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top = \mathbf{W}\mathbf{\Sigma}^2\mathbf{W}^\top \tag{2.3}$$

La descomposición en valores singulares es útil para demostrar los resultados en la diagonal de Var $[\mathbf{z}]$. Usando los SVD de X se expresar la varianza de X como:

$$\text{Var}[\mathbf{x}] = \frac{1}{m-1}\mathbf{X}^\top\mathbf{X} \tag{2.4}$$

$$= \frac{1}{m-1}\left(\mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top\right)^\top \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top \tag{2.5}$$

$$= \frac{1}{m-1}\mathbf{W}\mathbf{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top \tag{2.6}$$

$$= \frac{1}{m-1}\mathbf{W}\mathbf{\Sigma}^2\mathbf{W}^\top \tag{2.7}$$

Donde se produce que $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ debido a que $\mathbf{U}$ la matriz de valores singulares es ortogonal. Esto demuestra que la covarianza de z debe de ser los coeficientes de la diagonal.

$$\text{Var}[\mathbf{z}] = \frac{1}{m-1}\mathbf{Z}^\top\mathbf{Z} \tag{2.8}$$

$$= \frac{1}{m-1}\mathbf{W}^\top\mathbf{X}^\top\mathbf{X}\mathbf{W} \tag{2.9}$$

$$= \frac{1}{m-1}\mathbf{W}^\top\mathbf{W}\mathbf{\Sigma}^2\mathbf{W}^\top\mathbf{W} \tag{2.10}$$

$$= \frac{1}{m-1}\mathbf{\Sigma}^2 \tag{2.11}$$

donde esta vez se usará la propiedad de la descomposición en valores singulares $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$. Estos análisis indican que cuando se proyectan el dato x sobre z, con la transformación lineal $\mathbf{W}$, la representación del resultado tiene una matriz de covarianza (dada por $\mathbf{\Sigma}^2$), que inmediatamente implica que los elementos individuales z son mutuamente incorrelados. La principal propiedad de PCA es transformar los datos en una representación donde los datos estan mutuamente incorrelados [35].

2.5 Análisis discriminante lineal

El análisis discriminante lineal (LDA) es una técnica muy común para los problemas de reducción de la dimensionalidad. Funciona como un paso de preprocesamiento para el aprendizaje automático y las aplicaciones de clasificación de patrones.

El objetivo de este algoritmo es similar al anteriormente redactado, pero entre estos dos algoritmos existe

una gran diferencia. Como ya se ha explicado PCA es un algoritmo de aprendizaje no supervisado mientras que LDA es un algoritmo de aprendizaje supervisado.

El análisis discriminante lineal (LDA) busca los vectores en el espacio subyacente que consiguen una mejor discriminación entre clases (en lugar de los que mejor describen los datos). Esto quiere decir que dada una serie de características independientes con las cuales se describen los datos, LDA crea una combinación lineal de estas que produce las mayores diferencias de medias entre las clases deseadas [37].

El objetivo entonces de la técnica LDA es proyectar la matriz de datos original en un espacio dimensional inferior. Para ello se deben realizar tres pasos [43]:

1. Calcular la separabilidad entre diferentes clases (es decir, la distancia entre las medias de diferentes clases), lo que se denomina *varianza entre clases matriz entre clases o between-class variance between-class matrix.*
2. Calcular la distancia entre la media y las muestras de cada clase, que se denomina la *varianza dentro de la clase o matriz dentro de la clase o within-class*
3. Construir el espacio dimensional inferior que maximiza la varianza entre clases y minimiza la varianza dentro de la clase.

La *within-class variance* está definida por:

$$S_w = \sum_{j=1}^{c} \sum_{i=1}^{N_j} \left(\mathbf{x}_i^j - \mu_j\right) \left(\mathbf{x}_i^j - \mu_j\right)^T \tag{2.12}$$

donde $\mathbf{x}_i^j$ es $i-sima$ muestra de la clase j, μ_j es la media de la clase j, c es el número de clases y N_j es el número de muestras en la clase j

La llamada *between-class matrix* se obtiene según la siguiente ecuación:

$$S_b = \sum_{j=1}^{c} \left(\mu_j - \mu\right) \left(\mu_j - \mu\right)^T \tag{2.13}$$

donde μ representa la media de las clases .

El objetivo principal a realizar por LDA entonces es maximizar la medida entre clases y minimizar las medidas dentro de la clase. Para ello, se debe construir un espacio de menor dimensión. Después de calcular between-class variance (S_B) y la within-class variance (S_W), la matriz transformada $\mathbf{W}$ obtenida por el algoritmo LDA se puede calcular por el *Criterio de Fisher*:

$$\arg\max_W \frac{\mathbf{W}^T S_B \mathbf{W}}{\mathbf{W}^T S_W \mathbf{W}} \tag{2.14}$$

Y puede ser reformulada de la siguiente forma:

$$S_W \mathbf{W} = \lambda S_B \mathbf{W} \tag{2.15}$$

donde λ representan los autovalores de la matriz de transformación $(\mathbf{W})$. La solución de este problema puede ser resuelta calculando los autovalores $(\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_M\})$ y autovectores $(V = \{v_1, v_2, \ldots, v_M\})$ de $\mathbf{W} = S_W^{-1} S_B$, si S_W es no singular.
Los autovectores representan las direcciones del nuevo espacio, y los autovalores representan el factor de escala, la longitud o la magnitud de los autovectores. La dimensión de la matriz original $(\left(\mathbf{X} \in \mathbb{R}^{N \times M}\right))$ se verá reducida haciendo que se proyecte en la dimensión menor del espacio de LDA $\left(V_k \in \mathbb{R}^{M \times k}\right)$ como se puede observar en la siguiente ecuación:

$$Y = \mathbf{X} V_k \tag{2.16}$$

La dimensión de los datos después de la proyección es k ; por lo tanto, las características $M-k$ se ignoran o eliminan de cada muestra. Por lo tanto, cada muestra (x_i) que se representó como un punto en un espacio M-dimensional se representará en un espacio k-dimensional proyectándolo en el espacio dimensional inferior (V_k) como sigue,$y_i = x_i V_k$ [43].

Un buen ejemplo para poder visualizar las diferencias entres PCA y LDA es el siguiente:

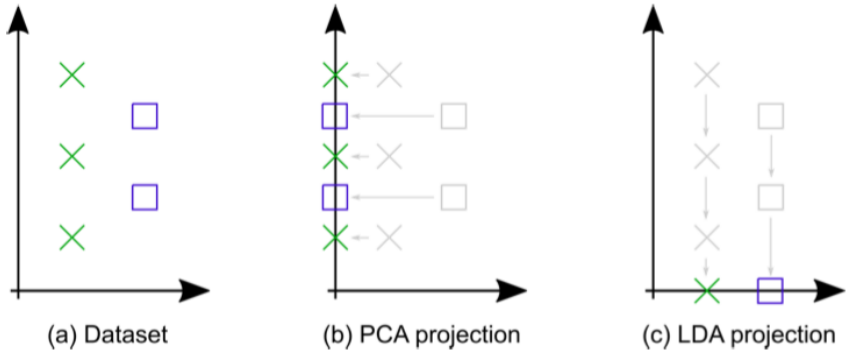

Figura 2.2 Ejemplo del distinto comportamiento de LDA y PCA en el mismo conjunto de datos [27].

El conjunto de datos está representado por la figura (a). En (b) se muestra el conjunto de datos proyectado sobre la base de que el PCA elegiría, ya que esto maximiza la varianza, independientemente de las etiquetas de clase (colores azul y verde en este caso). En (c) se muestra el conjunto de datos proyectado sobre la base que el LDA elegiría, ya que esto maximiza la separación de clases, en este caso las entradas de verde y azul [27].

2.6 Random Forest

El algoritmo de *Random Forest* pertenece al grupo de clasificadores agregados independientes. Un clasificador agregado independiente es un tipo de clasificador cuya principal característica es que el resultado se calcula según las predicciones de un set de clasificadores aislados los unos de los otros. Además, se realizan multiples arboles de decisión. Estos árboles se entrenan de forma independiente y las predicciones de los árboles se combinan a través del promedio [31].
Cada árbol es construido usando el siguiente algoritmo [18]:

1. Sea N el número de casos de una prueba y M el número de variables que se van a utilizar en el clasificador.

2. Sea m el número de variables de entrada a ser usado para determinar la decisión en un nodo dado; m debe ser mucho menor que M.

3. Se elige un conjunto de entrenamiento para este árbol y se usa el resto de los casos de prueba para estimar el error.

4. Para cada nodo del árbol, elegir aleatoriamente m variables en las cuales basar la decisión. Calcular la mejor particioón a partir de las m variables del conjunto de entrenamiento.

Así funcionaría este algoritmo visto gráficamente:

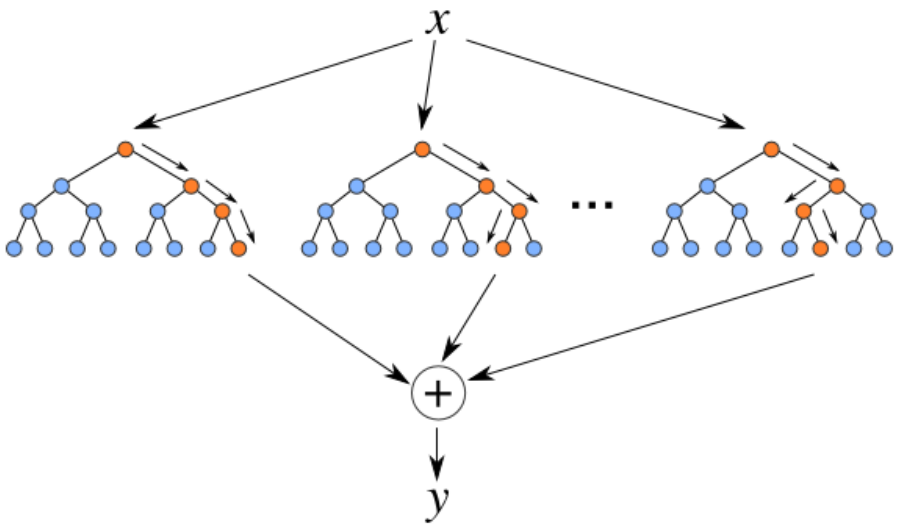

Figura 2.3 Algoritmo de *random forest* [29].

2.7 K vecinos más cercanos

Este algoritmo, también denominado *K-Nearest Neightbors* (KNN) es un método de aprendizaje supervisado cuya clasificación de resultados depende de la geometría de los datos, es decir, es un método de clasificación geométrico. Clasifica las nuevas instancias como la clase mayoritaria entre los k vecinos mas cercanos de los datos de entrenamiento.
El funcionamiento de este algoritmo para un set de datos de entrada $D = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_N, c_N)\}$. Donde D es un fichero de N casos, cada uno de los cuales está caracterizado por n variables que se usan para realizar la predicción, $X_1, \ldots, X_n$ y una variable a predecir, la clase C.. Los N casos se denotan por:

$$\begin{aligned} &(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_N, c_N) \text{ donde} \\ &\quad \mathbf{x}_i = \left(x_{i,1} \ldots x_{i,n}\right) \text{ para todo } i = 1, \ldots, N \\ &\quad c_i \in \left\{c^1, \ldots, c^m\right\} \text{ para todo } i = 1, \ldots, N \end{aligned} \tag{2.17}$$

donde entonces, $c^1, \ldots, c^m$ expresan los m posibles valores de la clase variable C. y el nuevo caso a clasificar se denomina $\mathbf{x} = (x_1, \ldots, x_n)$.
Funcionamiento del algoritmo con los datos anteriormente comentados [25]:

1. Para todo objeto ya clasificado (x_i, c_i), calcular su distancia euclídea $d_i = d(\mathbf{x}_i, \mathbf{x})$, siendo la formula de la distancia euclídea la siguiente:

$$d(X,Y) = \sqrt{\sum_{i=1}^{N} (y_i - x_i)^2} \tag{2.18}$$

2. Ordenar $d_i (i = 1, \ldots, N)$ en orden ascendente.

3. Quedarnos con los K casos D_x^K ya clasificados más cercanos a x

4. Asignar a $\mathbf{x}$ la clase más frecuente en $D_{\mathbf{x}}^K$

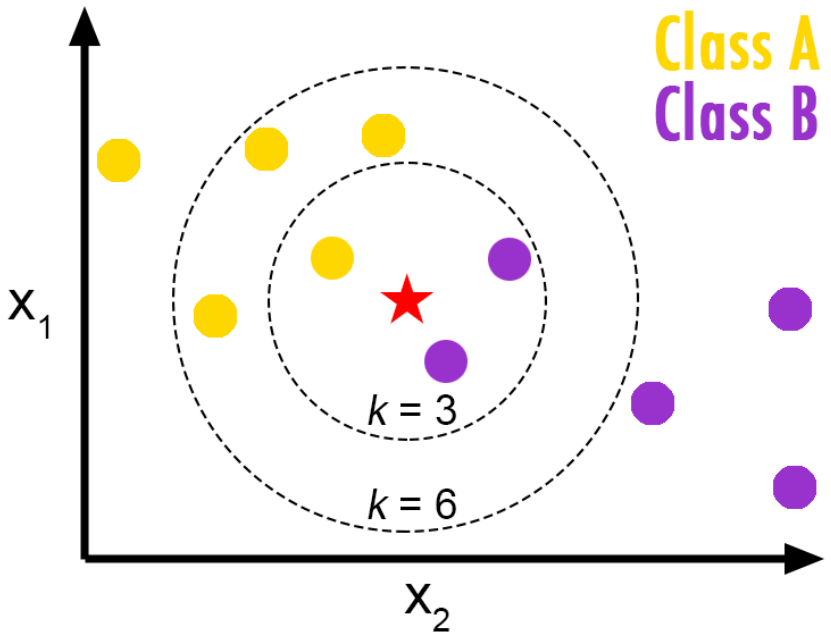

Figura 2.4 Ejemplo de clasificación del método de *k vecinos más cercanos* [8].

En este ejemplo, se puede observar que hay dos clases. Siendo la estrella roja el valor a predecir, con k=3 se diría que pertenece a la clase B, mientras si k=6, se predeciría que es de la clase B [8].

2.8 Redes Neuronales

Las redes neuronales forman parte de una tecnología que está en auge. Pueden ayudar visiblemente en una gran cantidad de aplicaciones. Su principal característica es su semejanza al funcionamiento biológico del cerebro humano [26]. Una neurona humana está formada por varias partes:

- Cuerpo (o soma)
- Dendritas
- Axón (o cilindro eje)

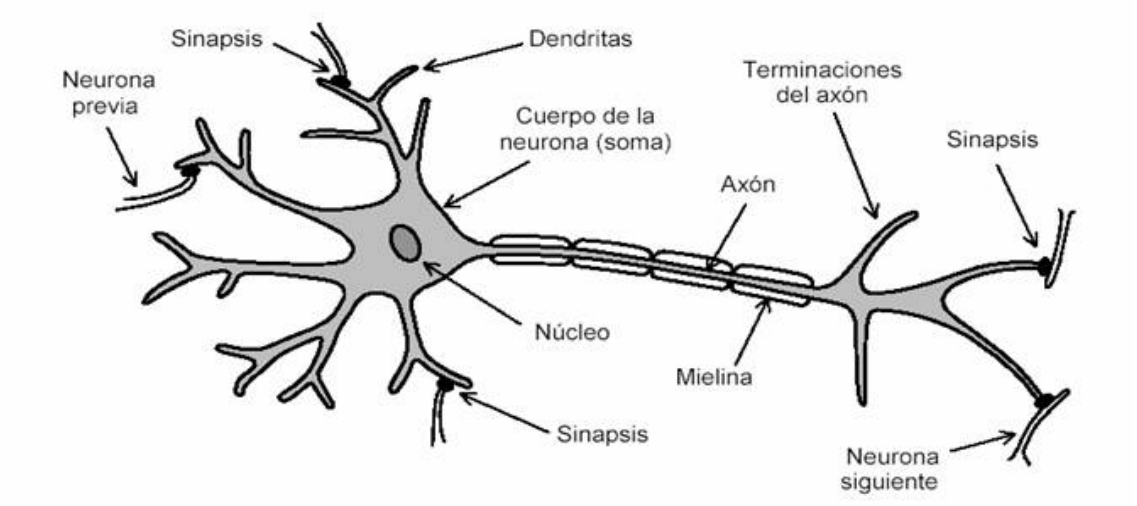

Figura 2.5 Partes de la neurona [2].

Las dendritas reciben señales de las neuronas adyacentes y son transmitidas al cuerpo en forma de potencial eléctrico. Estas señales eléctricas se integran por el soma . Si el potencial eléctrico es superior a un umbral, el cuerpo genera un corto impulso eléctrico. Este impulso se transmite por el axón que se ramifica y dirige el impulso a varias neuronas [26].
El primer modelo artificial de neurona fue establecido por McCulloch y Pitts en el año 1943. Este modelo recibe un conjunto de entradas $\{x_1, x_2, \ldots, x_D\}$ y devuelve una única salida y. Dentro de una ANN existen conexiones entre las diferentes neuronas por las que están formadas. Estas conexiones hacen una simulación de las conexiones interneuronales que tiene el cerebro humano, y pueden ser de mayor o menor intensidad. En el caso de las redes neuronales artificiales (RNA) esta intensidad es determinada por los pesos. De este modo, cada entrada x_i de una neurona se encuentra afectada por un peso w_i.

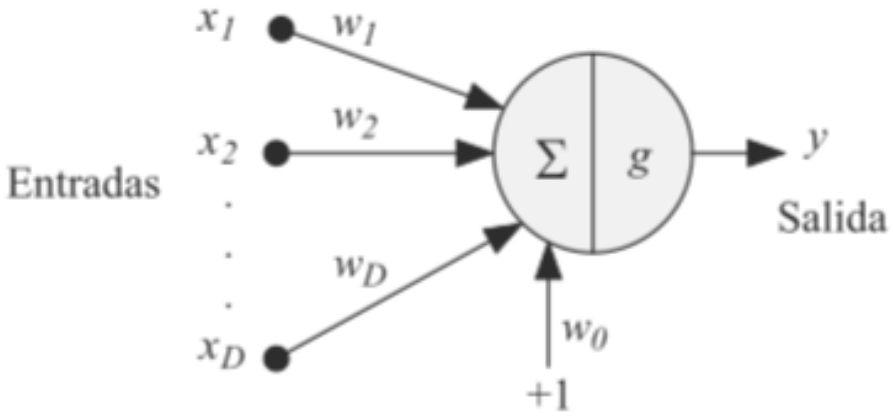

Figura 2.6 Modelo neuronal de McCulloch-Pitts [45].

2.8.1 Función de activación

Para obtener la salida y de la neurona se debe calcular la suma ponderada de las entradas, también llamada la activación de la neurona:

$$a = \sum_{i=1}^{D} w_i x_i + w_0 \tag{2.19}$$

Donde w_0 es el sesgo que se utiliza para hacer una compensación entre el valor medio de las entradas, sobre el conjunto entero de entrenamiento, y el valor medio de las salidas correspondientes. A continuación, a partir de este valor a se obtiene la salida y de la neurona mediante la función de activación:

$$y = g(a) = g\left(\sum_{i=1}^{D} w_i x_i + w_0\right) = g\left(\sum_{i=0}^{D} w_i x_i\right) \tag{2.20}$$

donde, se puede tratar el umbral w_0 como si fuera un peso más si se supone una entrada añadida x_0 con un valor fijo de 1.
La función de activación en el modelo de McCulloch-Pitts es la función denominada *escalón*:

$$g(a) = \begin{cases} 0 & \text{cuando} \quad a < 0 \\ 1 & \text{cuando} \quad a > 0 \end{cases} \tag{2.21}$$

Sin embargo, en los modelos actuales se utilizan otro tipo de funciones de activación como es el ejemplo de la *lineal*:

$$g(a) = a \tag{2.22}$$

la *exponencial*

$$g(a) = \frac{1}{1+e^{-a}} \tag{2.23}$$

la *tangente hiperbólica*

$$g(a) = \exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right) \tag{2.24}$$

o las que se han utilizado en este proyecto, la función *rampa o ReLu*

$$g(a) = \text{máx}(0,a) = \begin{cases} 0, & a < 0 \\ a, & a \geq 0 \end{cases} \tag{2.25}$$

y la función *sigmoidea*

$$g(a) = \tanh = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2.26}$$

La topología de la red neuronal se define como el número de neuronas que forman una RNA y la disposición en la que están conectadas.

2.8.2 Proceso de aprendizaje

Las RNA reciben un conjunto de datos de entrada que son transformados para producir una salida, con el fin de clasificarlos. Se puede ajustar nuestra red neuronal escogiendo valores adecuados para los parámetros que son ajustables (número de neuronas, número de capas, pesos, funciones de activación etc). Estos parámetros son el medio que emplea la red para almacenar su conocimiento sobre el problema a resolver.
Tras un proceso de entrenamiento se almacena en la RNA este conocimiento. Al igual que el aprendizaje humano, el de las redes neuronales está basado en utilizar ejemplos que van a representar el problema. Este conjunto es conocido como *conjunto de entrenamiento*. El objetivo del aprendizaje no es memorizar las relaciones entrada/salida sino modelar el proceso que ha generado estos datos. Así, una vez entrenada la RNA, la red es capaz de tanto manejar los datos de entrenamiento como datos distintos de los primeros, y aún así que no se incremente su error. Esto se llama *capacidad de generalización de la red*.
En este proceso se tiene como objetivo minimizar el error. En nuestro caso, la función de error que se ha utilizado es la del *Error Cuadrático Medio o ECM*:

$$E = \frac{1}{N}\sum_{n=1}^{N}(y^n - t^n)^2 \tag{2.27}$$

Esta función permite que las salidas de la red puedan modelar la función de distribución media de las salidas deseadas usadas durante el aprendizaje.

2.8.3 Redes de una capa: El perceptrón

El perceptrón es la RNA más sencilla. Está compuesta por una capa de neuronas a la entrada y otra capa de neuronas de salida.

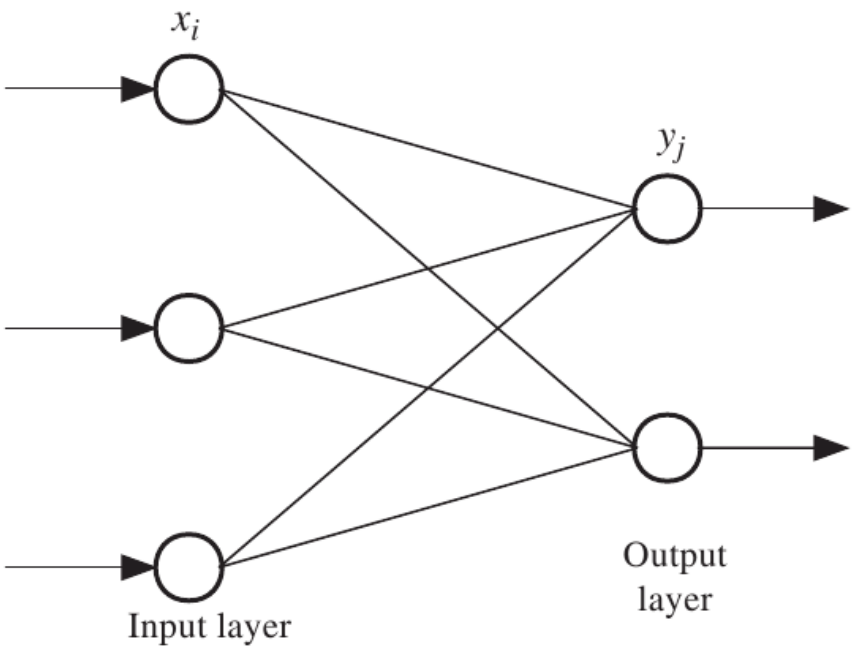

Figura 2.7 Perceptrón de una capa (Hsieh, 2009)[39].

La salida del perceptrón se calcula con [45]:

$$y = g(a) = g\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right) = g\left(\mathbf{w}^T \phi\right) \tag{2.28}$$

Donde ϕ es el vector formado por las funciones de activación $\phi_0, \ldots, \phi_M$ y $\phi_0 = 1$ también se tiene que tener en cuenta que g es la función escalón (Ecuación 2.14) definida para los valores $\{-1,1\}$.
El perceptrón adapta sus pesos en función del error entre la salida que tiene la red y la salida que se desea obtener. El perceptrón es un método de aprendizaje supervisado. Para explicar este método, se va a realizar un problema de clasificación entre dos clases en el que a cada vector de entrada x^n se le asocia un valor objetivo t^n de modo que si x^n pertenece a la clase $\mathscr{C}_1$ la salida deseada de la red es $t^n = 1$ y si x^n pertenece a la clase $\mathscr{C}_1$ la salida de la red es $t^n = -1$. Considerando que cada vector de entrada x^n genera su correspondiente vector de activación ϕ^n y teniendo en cuenta las ecuaciones (2.19) y (2.14) se obtiene que, para que el perceptrón consiga una clasificación correcta, se debe cumplir que $\mathbf{w}^T \boldsymbol{\phi}^n > 0$, si $\mathbf{x}^n \in \mathscr{C}_1, \mathbf{y}\mathbf{w}^T \boldsymbol{\phi}^n < 0$, si $\mathbf{x}^n \in \mathscr{C}_2$, es decir, $\left(\mathbf{w}^T \phi^n\right) t^n > 0$. Se debe minimizar pues, la siguiente función denominada el *criterio del perceptrón*:

$$E^{perc}(\mathbf{w}) = -\sum_{\phi^n \in \mathscr{M}} \left(\mathbf{w}^T \phi^n\right) t^n \tag{2.29}$$

donde $\mathscr{M}$ es el conjunto de vectores ϕ^n que no se han clasificado correctamente.
Cada vez que un patrón de entrada x^n es clasificado incorrectamente, si $\mathbf{x}^n \in \mathscr{C}_1$ los pesos se incrementan, y cuando se clasifican correctamente, los pesos se incrementan de acuerdo con la regla denominada *regla de aprendizaje del perceptrón simple:*

$$w_j(\tau + 1) = w_j(\tau) + \eta \phi_j^n t^n \tag{2.30}$$

donde η representa la tasa de aprendizaje.

2.8.4 El perceptrón multicapa

La gran cantidad de limitaciones de las redes de una sola capa hizo que se implementasen redes con capas ocultas entre la capa de entrada y la de salida. Este tipo de arquitectura se denomina perceptrón multicapa o MLP (*MultiLayer Perceptron*) y ha sido implementado en este proyecto para realizar el reconocimiento automático de instrumentos:

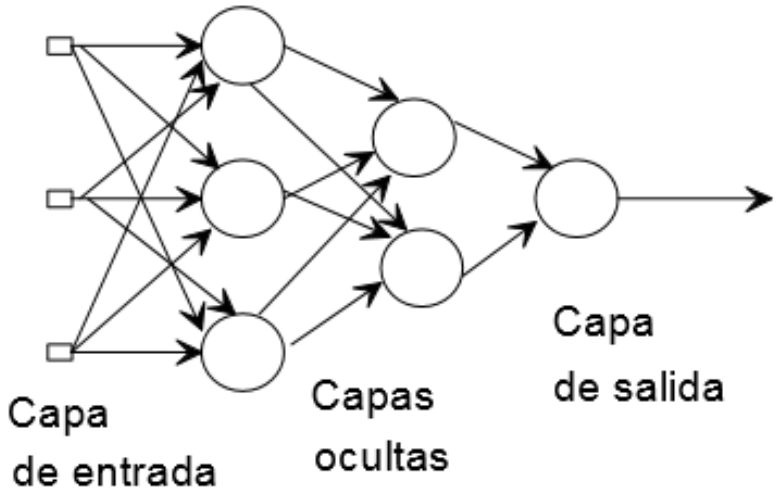

Figura 2.8 Perceptrón de multicapa [40].

Entonces MLP consta de D entradas, M neuronas en su capa oculta y C unidades de salida. El nivel de activación a_j de la neurona j de la capa oculta se calcula aplicando una combinación lineal de las D entradas de x_i que recibe sobre la que, se aplica una función de activación g y se obtiene la salida z_j de dicha neurona:

$$z_j = g\left(a_j\right) = g\left(\sum_{i=0}^{D} w_{ji} x_i\right); k = 1,2,\ldots,M \tag{2.31}$$

donde w_{ji} es el peso asociado a la neurona j y a la entrada x_i. Cada salida de la red se obtiene como una suma ponderada de las salidas de las neuronas de la capa oculta, sobre la que se aplica una función de activación. La salida de la neurona k será entonces:

$$y_k = \tilde{g}\left(a_k\right) = \tilde{g}\left(\sum_{j=0}^{M} w_{kj} z_j\right) = \tilde{g}\left(\sum_{j=0}^{M} w_{kj} g\left(\sum_{i=0}^{D} w_{ji} x_i\right)\right); k = 1,2,\ldots,C \tag{2.32}$$

Las funciones de transferencia g y $\overline{g}$ no tienen porque ser iguales, por ejemplo en nuestro caso, se utilizan funciones *ReLu*, *sigmoidea* y *tangente hiperbólica*

Algoritmo de retropropagación del error

El MLP, al igual que el perceptrón simple, basa el aprendizaje de sus pesos según una regla de ajuste del error. El algoritmo utilizado es el método de descenso del gradiente con el fin de ajustar los pesos de la red neuronal. Este ajuste se realiza comenzando por la capa de salida según el error cometido y se va propagando hacia las capas anteriores, hasta llegar a las capas de entrada. Este algoritmo se denomina *algortimo de retropropagación del error o backpropagation algorithm*. Esta formado por dos fases que se repiten hasta conseguir un error mínimizado [45]:

- La primera fase consiste en aplicar un patrón a las entradas de la red y que su efecto se propague a través de la misma capa a capa. Al final, la red presenta un conjunto de salidas en respuesta a dicho patrón de entrada. Para un conjunto de pesos dado se le aplica la ecuación (2.25). El valor inicial del conjunto de pesos en la primera iteración suele ser pequeño y aleatorio.

- En la segunda fase, los pesos se recalculan según una regla de ajuste del error. Se calcula el valor de la función de error la cual compara la respuesta actual y la respuesta deseada, posteriormente, este error se propaga hacia atrás.

A continuación se va a explicar en detalle la segunda fase. Se tiene en cuenta que en este proyecto se ha utilizado el error cuadrático medio, por lo tanto para cada patrón $\mathbf{x}^n$ se obtiene la siguiente función de error:

$$E^n = \frac{1}{2}\sum_{k=0}^{C}(e_k^n)^2 \tag{2.33}$$

el factor $1/2$ se introduce para facilitar el cálculo y no altera el resultado de la función. El error e_k^n se define como:

$$e_k^n = y_k^n - t_k^n \tag{2.34}$$

Supongamos que estamos en la iteración n del algoritmo de aprendizaje en el que se ha introducido como entrada $\mathbf{x}^n$. El algoritmo de retropropagación corrige el peso w_{kj} añadiéndole un incremento Δw_{kj}. Este incremento es proporcional al gradiente $\partial E/\partial w_{kj}$ y determina la dirección del espacio para obtener el valor del peso w_{kj} que conduce a un mínimo de E. Según la regla de la cadena para derivadas parciales, el gradiente se representará como:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = -e_k\tilde{g}'(a_k)z_j \tag{2.35}$$

esto se obtiene al derivar las ecuaciones (2.25), (2.26), (2.27) y $a_k = \sum_{j=0}^{M} w_{kj}z_j$ con respecto a las variables indicadas:

$$\frac{\partial E}{\partial e_k} = e_k \qquad \frac{\partial e_k}{\partial y_k} = -1 \qquad \frac{\partial y_k}{\partial a_k} = \tilde{g}'(a_k) \qquad \frac{\partial a_k}{\partial w_{kj}} = z_j \tag{2.36}$$

Teniendo en cuenta que *gradiente local* δ de una neurona k es definido por:

$$\delta_k = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial a_k} = -e_k\overline{g}'(a_k) \tag{2.37}$$

se obtiene que el incremento aplicable al peso w_{kj} es:

$$\Delta w_{kj} = \eta\frac{\partial E}{\partial w_{kj}} = -\eta\,\delta_k z_j \tag{2.38}$$

donde η es el ratio de aprendizaje. Esta ecuación señala que, para encontrar el incremento necesario, solo hay que multiplicar el valor de δ para la unidad de salida de la neurona por el valor z de la unidad de entrada [45]. Los pesos de la capa de salida se ajustan según la siguiente expresión [28]:

$$w_{k,j}^{y}(n+1) = w_{kj}^{y}(n) + \eta\,\delta_j^{y} z_k(n) \tag{2.39}$$

Sin embargo, si la neurona k se encuentra en en la capa oculta no existe un modo directo para calcular su error, ya que no se sabe la salida deseada en ese punto. Es por esto, que el error se debe calcular recursivamente teniendo en cuenta todas las neuronas de la capa de la salida a la que va conectada dicha neurona k. Se define como j a la neurona de la capa oculta a la que se quiere hacer una aproximación del gradiente, mientras que k es cada una de las neuronas de salida. El gradiente local δ_j se define para una neurona en la capa oculta como:

$$\delta_j = -\frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial a_j} = -\frac{\partial E}{\partial z_j}g'(a_j) \tag{2.40}$$

Se debe realizar de nuevo la regla de la cadena para calcular la primera derivada parcial:

$$\frac{\partial E}{\partial z_j} = \sum_k e_k\frac{\partial e_k}{\partial z_j} = \sum_k e_k\frac{\partial e_k}{\partial a_k}\frac{\partial a_k}{\partial z_j} = \sum_k e_k\tilde{g}'(a_k)w_{kj} = -\sum_k \delta_k w_{kj} \tag{2.41}$$

Sustituyendo en la ecuación (2.33):

$$\delta_j = g'(a_j)\sum_k \delta_k w_{kj} \tag{2.42}$$

El ajuste de los pesos de la capa oculta, viene dado por:

$$\Delta w_{ji} = \eta\,\delta_j x_i \tag{2.43}$$

o lo que es lo mismo:

$$w^{y}_{k,j}(n+1) = w^{y}_{kj}(n) + \eta\,\delta^{z}_{j} x_k(n) \tag{2.44}$$

El error calculado al introducir el patrón n-ésimo es un error individual. Entonces, los pesos se actualizan cada vez que un nuevo patrón $\mathbf{x}^n$ se introduce en la red. Este tipo de aprendizaje se denomina *aprendizaje estocástico*.

3 Aplicaciones de la inteligencia artificial en el análisis de datos musicales

En la música todos los sentimientos vuelven a su estado puro y el mundo no es sino música hecha realidad.

Arthur Schopenhauer

Este capítulo tiene como propósito exponer el *estado del arte* del panorama musical. En primer lugar, se debe comenzar hablando del MIR ya que es el mayor propulsor de investigación musical.

El MIR es una ciencia interdisciplinaria [19], cuyo principal objetivo es comparar los algoritmos y sistemas mas avanzados en el ámbito musical. Para ello, se organiza una ”competición” anual (MIREX o Music Information Retrieval Evaluation eXchange) que tendrá lugar este año en la 20ª Conferencia ISMIR (International Society for Music Information Retrieval Conference [1]. MIR actualmente es un pequeño pero creciente campo investigación que realiza muchas aplicaciones en negocios y centros académicos para manipular, categorizar o crear música [19].
Tanto MIREX como MIR están enfocados al tratamiento del audio, ya que la mayoría de los investigadores del MIR son expertos en el ámbito del procesamiento de la señal. Sin embargo, hay muchos desafíos a los que se enfrentan las investigaciones de MIREX y MIR, la mayoría tienen su origen en los problemas de propiedad intelectual que rodean a la música. La incapacidad actual que tienen los investigadores para probar sus experimentos con las colecciones de pruebas MIREX fuera de su ciclo anual está obstaculizando el desarrollo de sistemas MIR optimizados [32].

Algunas de las investigaciones de tecnologías más punteras son: la descripción de sonido y música, la recuperación de información musical, síntesis de la voz cantada, separación de fuentes de audio, procesamiento de audio y música. Para ello, se deben proponer algunos retos a superar [13]:

1. Aprovechar el carácter multimodal de la música para mejorar su procesamiento automático.
2. Reducir la brecha semántica entre las características automáticas y los descriptores de usuario.
3. Conectar descripción musical y creación musical.
4. Incorporar técnicas avanzadas de aprendizaje para el procesamiento de música.
5. Aplicar el conocimiento que se tiene actualmente de un modelado de voz de canto a un modelo basado en estadísticas de vocalizaciones de animales.

Además, en el MIREX como anteriormente se ha comentado, existen una gran cantidad de ”challenges” que se pretenden conseguir superar este año, 2019, como es el caso de:

- Clasificación de audio (Entrenamiento/Validación)
- Seguimiento de pulsos en un audio
- Extracción de la melodía de un audio

- Identificación de versiones de canciones
- Separación de la voz cantada
- Estimación y búsqueda de las multiples frecuencias fundamentales
- Clasificación por etiquetas de audio
- etc

A continuación se explicarán algunos de estos ”challenges” con el fin de exponer los principales avances que se quieren desarrollar este año.

Clasificación de audio (Entrenamiento/Validación)
Dividido en varias secciones, *Clasificación de audio (Entrenamiento/Validación)*, esta basado en la clasificación de música siguiendo un proceso de dos etapas: entrenamiento de modelos de clasificación utilizando datos etiquetados (training) y probar modelos utilizando datos nuevos o datos que no han sido vistos anteriormente (testing). El modelo a realizar es bastante similar al que se ha hecho en este proyecto. Sin embargo, estas secciones a clasificar son diferentes al reconocimiento automático de instrumentos. Estos subapartados son:

- Identificación de compositores de música clásica: Reconocimiento automático de los compositores de cada canción. Algunos de ellos son: Bach, Beethoven, Brahms, Chopin, Dvorak, Handel, Haydn, Mendelssohnn, Mozart, Schubert, Vivaldi.
- Clasificación de géneros POP estadounidense: Clasificación de audios según el tipo de género musical (Blues, Jazz, Country/Western, Baroque, Classical, Romantic, Electronica, Hip-Hop, Rock, HardRock/Metal.)
- Clasificación de géneros musicales latinos: Clasificación de audios según el tipo de género musical, pero con la diferencia de ser tipo de música latina: Axe, Bachata, Bolero, Forro, Gaucha, Merengue, Pagode, Sertaneja, Tango.)
- Clasificación de audios según el estado de ánimo: Este conjunto de datos requiere algoritmos para clasificar el audio según el estado de ánimo de la pista. Para ello se debe dividir en cinco clústeres:
  - Cluster 1: apasionado, entusiasta, confiado, bullicioso, ruidoso.
  - Cluster 2: alegre, divertido, dulce, amable, bondadoso.
  - Cluster 3: culto, conmovedor, melancólico, agridulce, otoñal, melancólico.
  - Cluster 4: humorístico, peculiar, ingenioso, irónico.
  - Cluster 5: agresivo, furioso, tenso, intenso, volátil, visceral.
- Clasificación de K-POP según el estado de animo: Se clasifican los audios según los clústeres anteriormente explicados pero las canciones pertenecen al género K-POP (música popular coreana)
- Clasificación de K-POP según el género musical: Se hace una clasificación de música popular coreana según su género (Ballad, Dance, Folk, Hip-hop, R&B, Rock, Tro.)

Seguimiento de pulsos en un audio
El objetivo de la tarea de seguimiento automático de los pulsos es rastrear cada ubicación de cada uno de ellos en una colección de archivos de audio. A diferencia de la tarea de extracción de tempo de audio, cuyo objetivo es detectar tempos para cada archivo, la tarea de seguimiento de pulsos tiene como objetivo detectar todas las ubicaciones de los pulsos en los audios. Los algoritmos se deben evaluar según la precisión de predicción de las ubicaciones de los pulsos anotadas por un grupo de oyentes.

Extracción de la melodía de un audio
El objetivo de la evaluación de extracción de melodía de los audios MIREX es identificar el contorno del tono de una melodía a partir de audio musical polifónico. El tono se expresa como la frecuencia fundamental de la voz melódica principal, y se informa en forma de trama en una cuadrícula de tiempo espaciada uniformemente. La tarea consta de dos partes:

1. Detección de audio (decidir si un período de tiempo particular contiene un ”tono de melodía” o no).

2. Detección de tono (decidir el tono de melodía más probable para cada intervalo de tiempo).

Identificación de versiones de canciones

Este reto requiere que los algoritmos identifiquen, para una pista de audio de referencia, otras grabaciones de la misma composición o composición similar denominadas ”cover”. Una ”cover” es una canción versionada por otro cantante que no es el compositor original.
Dentro de la colección de audios, hay tanto varias ”canciones originales” como composiciones ”canciones versionadas”. Además la base de datos contiene audios de una gran variedad de estilos musicales (por ejemplo, música clásica, jazz, gospel, rock, folk-rock, etc.)

Separación de la voz cantada

El objetivo principal que se tiene es separar la voz de un cantante de POP de los demás componentes de una canción como es el ejemplo de los instrumentos. Este reto es muy interesante porque se tiene que utilizar características que identifiquen tanto a una voz de una persona como los instrumentos que están de fondo en la canción. Para ello se debe utilizar algoritmos de aprendizaje automático definiendo la voz del cantante y los diferentes instrumentos que son típicos en una canción POP.

Estimación y búsqueda de las multiples frecuencias fundamentales

Una señal musical compleja puede representarse por los contornos de su frecuencia fundamental (F0) de sus fuentes constituyentes. Este es un concepto muy útil para la mayoría de los sistemas de recuperación de información musical. Ha habido muchos intentos de realizar una estimación múltiple (también conocida como polifónica) de las frecuencia fundamentales de un audio. El objetivo de la estimación y el seguimiento múltiples de frecuencias fundamentales es identificar los F0 activos en cada intervalo de tiempo y realizar un seguimiento continuo de las notas y timbres en una señal musical compleja. Para ello, se busca evaluar algoritmos de seguimiento y estimación de múltiples F0 de última generación. Dado que el seguimiento de las frecuencias fundamentales de todas las fuentes en una mezcla de audio compleja puede ser muy difícil, se puede dividir este problema a 3 casos:

1. Estimar las frecuencias fundamentales activas en una base fotograma a fotograma.
2. Seguimiento de los contornos de la nota en forma continua de tiempo. (como en audio-a-midi). Esta tarea también incluirá una subtarea de transcripción de audios de un piano.
3. Obtención del timbre de la pista en una base de tiempo continuo.

Clasificación por etiquetas de audio

Este objetivo está bastante relacionado con este proyecto. En la clasificación por etiqueta de audio se compararán las capacidades de varios algoritmos para asociar etiquetas descriptivas a clips de audio de 10 segundos (canciones). Se usan dos conjuntos de datos para implementar un par de subtareas, basadas en los conjuntos de datos de etiquetas de ”MajorMiner” y ”Mood”. Los algoritmos serán evaluados tanto por su capacidad para realizar clasificaciones binarias de etiquetas, como en su capacidad para clasificar etiquetas para una canción pidiéndoles que devuelvan una puntuación de afinidad para cada pareja de etiqueta/canción. Es una continuación de otro ”challenge” que se celebró en 2008 que tenía el mismo propósito así como en 2009 y se continuó desde el 2010 al 2014. La colección de datos más similar a este trabajo es la de *MajorMiner Tag Dataset*, aunque las etiquetas que se deben analizar en este ”challenge” son muchas más que en nuestro dataset. En nuestro caso, clasificamos seis instrumentos, mientras que aquí se intentan identificar tanto estilos musicales (pop, rock, techno, country, trance, etc) como tipo de instrumentos (trompeta, saxofón, guitarra, órgano, etc), teniendo un total de 73000 etiquetas diferentes, de las cuales 12000 han sido verificadas al menos por dos usuarios.
Al ser un objetivo MIREX 2019, no saldrán los resultados hasta el 2020 y como la última competición fue en 2014, a continuación se analizarán otras aplicaciones más actuales en el análisis de datos musicales [1].

En el ámbito del procesamiento de señales de audio se busca avanzar en la comprensión de las señales musicales mediante la combinación de métodos de procesamiento de señales y aprendizaje automático. Combinando estos enfoques de investigación, se puede abordar problemas prácticos relacionados con la exploración y la recomendación de la música, la clasificación de los sonidos y la evaluación de la interpretación musical para la educación.
Entre la música y aprendizaje automático se puede observar que existe un gran interés en modelar actuaciones musicales. Con machine learning, se pretende automatizar el proceso creativo de la manipulación de las

propiedades del sonido. A su vez, se estudia el uso de las emociones en las interfaces cerebro-computadora (música). Especialmente, se realizan investigaciones para el refuerzo de las interfaces cerebro-máquina y su capacidad para mejorar condiciones como la depresión, la enfermedad del Parkinson y el autismo. También se realizan estudios basados en cómo aprendemos los instrumentos musicales [13].

4 Explicación concreta de las herramientas, datos y algoritmos utilizados

Solo el cambio perdura

Heráclito

En este Capítulo se desarrollarán las herramientas que se han utilizado para la realización de los códigos, tanto para la extracción de características como para la parte de aprendizaje automático utilizando los algoritmos explicados en el Capítulo 2.

4.1 Herramientas

4.1.1 Lenguaje de programación

El lenguaje utilizado para realizar el código es *Python 2.7.15*, ya que es el lenguaje de programación más completo para el desarrollo de inteligencia artificial. Además, contiene una gran cantidad de librerías para el parseamiento de datos.

4.1.2 Librerias utilizadas

- **Essentia:** se trata de una librería creada por la UPF, de código abierto, cuya función principal en este proyecto ha sido la extracción de características de los audios de los distintos instrumentos. Para ello se ha utilizado el código de *Music Extractor* que devuelve un archivo ”.json” con todas las características de los audios a analizar [5].
- **NumPy:** es una librería que contiene funciones matemáticas de matrices y vectores, con lo cual ha sido de gran ayuda para el manejo de datos [14].
- **Matplotlib:** es una biblioteca de trazado 2D de Python utilizada para realizar gráficas [11].
- **Sklearn:** es una librería de python de código abierto que contiene algoritmos de machine learning [20].
- **Keras:** es una API de redes neuronales de alto nivel, escrita en pyhton y capaz de ejecutarse cobre TensorFlow, CNTK o Theano. En este proyecto, se ejecuta sobre TensorFlow [9].
- **Pandas:** es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas, diseñadas para que el trabajo con datos ”relacionales” o ”etiquetados” sea fácil e intuitivo [15].
- **Seaborn:** es una biblioteca de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos [21].
- **Simplejson:** es una librería simple, rápida, completa y extensible que sirve para codificar y decodificar archivos ”.json” [22].

- **Python-csv:** es una herramienta de python para manipular archivos .csv. En este proyecto se ha utilizado para pasar los archivos ".json" a formato ".csv" [17].

4.1.3 Entorno

El entorno en el que se ha desarrollado el proyecto ha sido *Visual Studio Code*. Visual Studio Code es un editor de código fuente potente que se ejecuta en el escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros idiomas (como C ++, Java, Python, PHP, Matlab etc) y tiempos de ejecución (como .NET y Unity) [24].

4.2 Datos

Para que un algoritmo en Machine Learning funcione correctamente, es indispensable que los datos extraídos incluyan las características más relevantes para poder clasificar con una alta probabilidad de acierto los instrumentos. Para ello, se ha decidido usar las características que se van a explicar a continuación.

4.2.1 Características

MFCC

Los MFCC (Mel-Frequency Cepstral Coefficients) son el conjunto de coeficientes que representan el espectro de potencia a corto plazo del sonido. Se basa en una transformada de coseno lineal de un espectro de potencia logarítmica en una escala de Mel no lineal de frecuencia. MFCC se utiliza para extraer el vector de características de la onda de sonido. El algoritmo MFCC se basa en percepciones auditivas humanas y teniendo filtro basado en la escala de Mel. El proceso de extracción de características se explica en el diagrama de bloques dado [36]:

Figura 4.1 Diagrama de bloques para extraer los MFCCs [36].

1. Se fragmenta la señal y se pasa por una ventana Hamming :

$$w(n) = (1-a) - a\cos\frac{2\pi n}{M-1}, 0 \leq n \leq M-1 \tag{4.1}$$

2. Se le aplica la transformada discreta de fourier (DFT). La muestra de dominio de tiempo se convierte en dominio de frecuencia usando FFT (La transformada rápida de Fourier). FFT es un algoritmo eficiente de DFT. Usualmente realizamos FFT para obtener la respuesta de frecuencia de magnitud para cada cuadro.

$$Y(\omega) = FFT[h(t) * X(t)] = H(\omega) * X(\omega) \tag{4.2}$$

   Donde $X(\omega), H(\omega)$ y $Y(\omega)$ son las transformadas de Fourier de X (t), H (t) y Y (t) respectivamente. Convierte cada fragmento de K muestras del dominio del tiempo al dominio de la frecuencia.

3. El banco de filtros Mel está formado por filtros triangulares. Aquí, los filtros están espaciados a lo largo de la frecuencia de Mel, que está relacionada con la frecuencia lineal común f mediante la siguiente ecuación:

$$\text{F(Mel)} = 2595 * \log 10[1 + f/700] \tag{4.3}$$

   La frecuencia de mel es proporcional al logaritmo de la frecuencia lineal, reflejando efectos similares en la percepción auditiva subjetiva del humano.

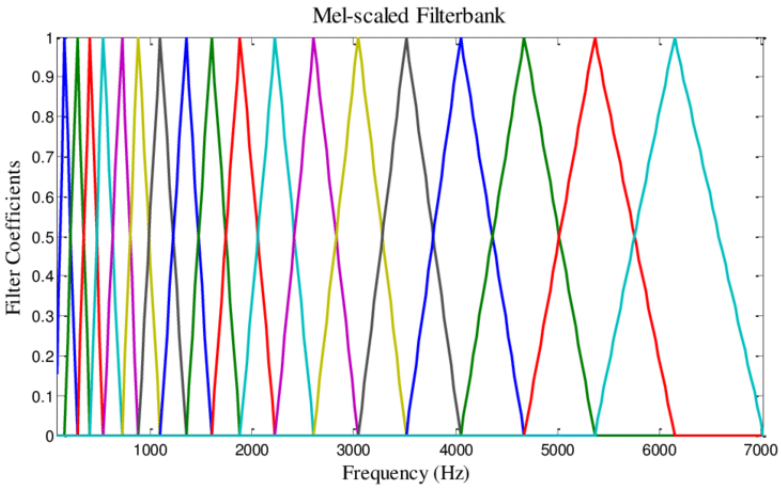

Figura 4.2 Ejemplo de bancos de filtros en la escala de Mel [38].

4. Aplicamos DCT en la energía Ek que se obtuvo como resultado de los filtros de paso de banda triangulares. El número de coeficientes viene dado por L en la fórmula siguiente:

$$C_m = \sum_{k=1}^{N} \cos\left[\frac{m \cdot (k-0{,}5) \cdot \pi}{N}\right] \cdot E_k, m = 1,2,\ldots,L \tag{4.4}$$

En nuestro caso, el algoritmo producido por Essentia hace una división con un banco de filtros de 40 bandas que van de 0 a 11000Hz. Después toma el valor logarítmico de la energía del espectro en cada banda mel. A continuación, realiza la transformada de coseno discreta a esas 40 bandas y halla 13 coeficientes [5]. Como características, se extrae la media y la varianza de cada uno de estos 13 coeficientes.

GFCC

Este algoritmo calcula los coeficientes cepstrales de frecuencia gammatono de un espectro. Esto es un equivalente de los MFCC anteriormente explicado, pero utilizando un banco de filtros de gammatono (ERBBands) escalado en una escala de Equivalent Rectangular Bandwidth (ERB).

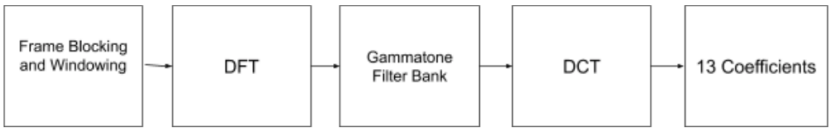

Figura 4.3 Diagrama de bloques para extraer los GFCCs [34].

El filtro utilizado es el siguiente:

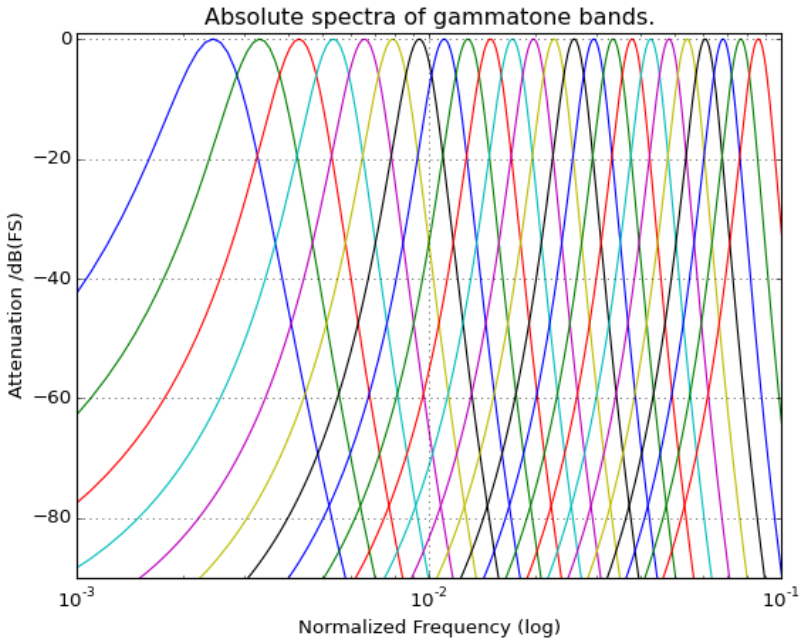

Figura 4.4 Ejemplo de bancos de filtros en la escala de Gammatono [7].

Para este proyecto, se ha extraído la media y la varianza de cada uno de los 13 coeficientes gammatono.

Centroide espectral

El centroide espectral es una medida que indica dónde está el ”centro de masa” del espectro. Está ligado al ”brillo” de un instrumento y, por lo tanto, se utiliza para caracterizar el timbre musical. Se calcula según la siguiente fórmula:

$$\text{Centroide espectral } = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)} \tag{4.5}$$

Donde $x(n)$ representa la magnitud de un número bin n, y $f(n)$ representa la frecuencia central de ese bin [23].
La característica extraída es la media.

Energía espectral

La energía espectral está definida por la siguiente ecuación:

$$E_s = \langle x(n), x(n) \rangle = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{4.6}$$

Se ha añadido la media de la energía espectral como característica.

Prominencia del tono de un espectro

La prominencia del tono de un espectro o también denominada *pitch salience*. La prominencia del tono viene dada por la relación entre el valor más alto de correlación del espectro y el valor de correlación no desplazada. Este algoritmo utiliza la autocorrelación en el ”espectro” de entrada y, por lo tanto, hereda sus requisitos y excepciones de entrada. El pitch salience viene dado por la siguiente formula:

$$\text{Pitch salience } = \frac{\text{máx}\,(r(n)) \text{ for } n \in [n_{\text{min}} n_{\text{max}}]}{r(0)} \tag{4.7}$$

Donde $n_{\text{mín}}$ y $n_{\text{máx}}$ están definidos por el análisis de rango del tono y $r(n)$ es la autocorrelación de la señal $x(n)$:

$$r(n) = \frac{1}{K} \sum_{k=0}^{K-n-1} x(k) \times x(k+n) \tag{4.8}$$

En este caso se va a utilizar la media de la prominencia de un tono del espectro [16].

Caída espectral

Este algoritmo calcula la frecuencia de caída de un espectro o *spectral roll-off*. La frecuencia de roll-off se define como el punto de reducción espectral en la frecuencia, de modo que el 95 % de la energía de la señal se encuentra debajo de esta. Se correlaciona de alguna manera con la frecuencia de corte armónico/ruido. La frecuencia de reducción se puede utilizar para distinguir entre sonidos armónicos (por debajo del roll-off) y sonidos ruidosos (por encima de roll-off).

$$\sum_{0}^{fc} a^2(f) = 0.95 \sum_{0}^{sr/2} a^2(f) \tag{4.9}$$

Donde f_c es la frecuencia de la caída espectral , $sr/2$ es la frecuencia de Nyquist [41]. Para la extracción de características, se ha utilizado la media de la caída espectral.

Cutorsis espectral

La curtosis caracteriza la función de distribución de una variable aleatoria. Da una medida de la forma, es decir, cómo de plana o picuda es distribución alrededor de su valor medio. Se obtiene según el cuarto momento estandarizado:

$$m_4 = \int (x-\mu)^4 \cdot p(x) \quad \delta x \tag{4.10}$$

La curtosis (γ_2), se puede definir entonces como: $\gamma_2 = \frac{m_4}{\sigma^4}$
Se pueden obtener 3 tipos de distribución según el valor de la curtosis:

- $\gamma_2 = 3$ para una distribución normal o gaussiana.
- $\gamma_2 < 3$ para una distribución ”plana”.
- $\gamma_2 > 3$ para una distribución ”picuda”.

En nuestro caso, se ha decidido utilizar la media de la curtosis espectral [41].

Flujo espectral

Este algoritmo calcula el flujo espectral de un espectro. El flujo se define como la norma L2 de la diferencia entre dos cuadros o *frame* consecutivos del espectro de magnitud. Los marcos deben ser del mismo tamaño para obtener un resultado significativo.

$$S_{fux}^{(k)} = \sum_{m=m_l}^{m_u} \left(\frac{X^{(k)}(m)}{\mu_k} - \frac{X^{(k-1)}(m)}{\mu_{k-1}} \right)^2 \tag{4.11}$$

En nuestro caso, como es L2 se obtiene que los coeficientes de normalización μ_k y μ_{k-1}:

$$\mu_k^{(L_2)} = \sqrt{\sum_{m=m_l}^{m_u} \left(X^{(k)}(m)\right)^2} \tag{4.12}$$

En caso de no utilizar la normalización en el flujo se denominará *flujo espectral no normalizado* [33]. En este proyecto, se ha seleccionado la media del flujo espectral como característica a utilizar.

Sesgo espectral

El sesgo espectral o *spectral skewness* da la medida de la asimetría de una distribución alrededor de la media. Se computa según el momento de tercer orden.

$$m_3 = \int (x-\mu)^3 \cdot p(x) \quad \delta x \tag{4.13}$$

El sesgo (γ_1) está definido entonces por: $\gamma_1 = \frac{m_3}{\sigma^3}$ Por lo tanto se describe el nivel de asimetría de la distribución. Existen tres casos distintos:

- $\gamma_1 = 0$ indica que es una distribución simétrica.
- $\gamma_1 < 0$ indica que hay más energía en la parte derecha.

- $\gamma_1 > 0$ indica que hay más energía en la parte izquierda.

La caracterísitica utilizada para describir el sesgo ha sido la media [41].

Media cuadrática espectral

La media cuadrática espectral o *spectral RMS* para un set de N valores $\{x_1, x_2, \ldots, x_N\}$ de una variable discreta tiene la siguiente formula:

$$x_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_N^2}{N}} \tag{4.14}$$

Como en nuestro caso la señal que tenemos que analizar es un array que contiene un conjunto de N valores, se puede aplicar la formula 4.14 anteriormente descrita. En este caso, también se ha cogido la media para nuestra colección de características [12].

Tabla resumen de características utilizadas

Anteriormente se han ido explicando cuales son las características que se han extraído de cada una de las muestras de audio. Por ello, se va a utilizar una tabla para concluir esta sección a modo de resumen.

Tabla 4.1 Tabla resumen de características y parámetros usados.

Caracterísitcas	Parámetros
MFCC	Media y varianza cada uno de los 13 coeficientes
GFCC	Media y varianza de cada uno de los 13 coeficientes
Centroide espectral	Media
Energía espectral	Media
Prominencia espectral del tono	Media
Caída espectral	Media
Curtosis espectral	Media
Flujo espectral	Media
Sesgo espectral	Media
Valor cuadrático medio espectral	Media

Hacen un total de 60 parámetros por cada audio analizado.

4.2.2 Base de datos

La base de datos utilizada para el reconocimiento de instrumentos contiene 116 muestras de audio del clarinete, 132 muestras de la guitarra, 120 audios de oboe, 117 muestras de piano eléctrico, 127 del saxofón y 121 muestras de violin. A cada uno de los audios pertenecientes a esta BBDD, se les ha extraído las características anteriormente comentadas y posteriormente se les ha etiquetado según el instrumento al que pertenecen. Obteniendo así una base de datos compuesta por 733 audios con 60 características cada uno y etiquetados.

4.3 Algoritmos

En esta parte se van a describir los algoritmos que se han utilizado para la realización del proyecto.

4.3.1 Extracción de características

La extracción de características está dividida en dos partes:

1. **Extraer todas las características de los audios:** En esta primera parte se ha utilizado la librería *Essentia* y un código desarrollado por la UPF denominado ”Music extractor” [3]. Para poder utilizar este código, primero se han tenido que dividir los audios en carpetas separadas según el instrumento. A continuación, por terminal se ejecuta la siguiente orden:
   python2 music_extractor.py -d <DIR_INSTRUMENTO> –output-json INSTRUMENTO.json
   El resultado es un archivo *”.json”* el cual contiene todas las características de los audios que estaban en cada carpeta, es decir, si por ejemplo tenemos la carpeta ”VIOLIN” que contiene 121 muestras de

audio, este programa devolverá un archivo denominado ”violin.json” con todas las características que están en la página web de Essentia [4].

2. **Extraer los parámetros de la sección (4.2.1) :** A partir del código del Anexo (7.1.1) se extraen los parámetros anteriormente mencionados y posteriormente se guardan en un archivo *”.csv”* con el fin implementarlo posteriormente en el código (7.1.2) como entrada para las redes neuronales. Para ello, en primer lugar se ha pasado el archivo ”.json” a diccionario de python con la orden *json.load(”instrumento.json”)*, posteriormente se extraen las características de la sección 4.2.1, se etiqueta cada audio según el instrumento que es y por último se reescribe como un archivo de extensión ”.csv”.

4.3.2 Reconocimiento de instrumentos

En esta sección se va a desarrollar la parte de algoritmos utilizados para la clasificación de instrumentos. En el capítulo 2 se ha explicado analíticamente como funcionan los algoritmos de Machine Learning. Sin embargo, aquí se desarrollará cuales son los parámetros modificables de estos y en el siguiente capítulo se propondrá una batería de pruebas con el fin de poder comparar los resultados de los algoritmos.
El set de datos se va a dividir en dos partes: características y etiqueta de instrumento. Mientras que las características (se denominan ”X”) y son un total de 60, los instrumentos son la etiqueta final y se les llama ”Y”. Se realiza una subdivisión de estos dos conjuntos, el primero se llama *entrenamiento* y el segundo subconjunto se denomina *validación o prueba*. El primer subconjunto es utilizado para realizar el entrenamiento de cada algoritmo y debe tener un tamaño mayor que el set de validación, siendo sus proporciones aproximadamente 80% entrenamiento y 20% validación. El subconjunto de validación debe realizar pruebas para comprobar los resultados del entrenamiento.
Después de dividir el conjunto de datos, existen dos opciones para llevar a cabo el reconocimiento de instrumentos:

1. Minimizar el dimensionamiento de la matriz de características con los algoritmos PCA o LDA y posteriormente utilizar un algoritmo de clasificación automática.
2. Dejar el set de datos con las dimensiones que se tiene originalmente y a continuación usar un algoritmo de aprendizaje automático.

En el siguiente apartado se va a desarrollar la implementación de la disminución de dimensión en el código realizado en este proyecto.

Minimizar dimensionamiento de la matriz de características

1. **PCA**: El desarrollo matemático de este algoritmo se puede ver en la sección 2.4. En primer lugar se debe seleccionar el número de componentes va a tener la nueva matriz características y posteriormente se introducen solamente las muestras de entrenamiento de las características, es decir, *X_train* al algoritmo PCA ya que como se ha explicado anteriormente, PCA es un algoritmo de aprendizaje no supervisado. Por último, se tiene que aplicar la asignación (transformación) tanto al conjunto de entrenamiento *X_train*como al conjunto de pruebas *X_validation* para conseguir la dimensión deseada de todos los parámetros.
2. **LDA**: El análisis matemático de este algoritmo está en 2.5. El funcionamiento es similar a PCA, pero como se ha explicado anteriormente, LDA es un algoritmo de aprendizaje supervisado, por lo tanto cuando se ajusta el modelo, debe ser con todo el conjunto de entrenamiento *X_train e y_train*. Por último se debe hacer la transformación de las pruebas de validación de las características *X_validation*. La implementación de código de LDA se puede ver en el Anexo 7.1.3 y 7.1.4.

Hay que tener en cuenta que minimizar la matriz de características no tiene porqué estar correlacionado con la obtención de mayor probabilidad de acierto en el reconocimiento de instrumentos.

Clasificación automática de instrumentos

Esta sección se puede realizar haciendo la disminución de la matriz de características o sin hacerla. Se va a explicar los tipos de algoritmos cuya funcionalidad principal es el reconocimiento de clases de instrumentos.

1. **Random forest**: El algoritmo está explicado matemáticamente en (2.6). Se debe crear un clasificador con la profundidad con la que se desee tener el bosque aleatorio y a continuación se ajusta el modelo con las todos los datos de entrenamiento (*X_train e Y_train*). Por último se hace la predicción.

2. **KNN**: Este algoritmo se desarrolla en la sección (2.7). Al igual que en la *random forest* se debe crear un clasificador pero el parámetro a utilizar es el número de ”vecinos”. Posteriormente, se hace un ajuste del modelo con las todos los datos de entrenamiento (*X_train e Y_train*) y finalmente se realiza la predicción.
3. **Redes neuronales**: El análisis matemático de cómo funcionan las redes neuronales se puede ver en (2.8). Como ya se había comentado antes, el modelo de una red neuronal tiene varios parámetros modificables:
   a) Número de capas que tiene la red neuronal.
   b) Número de neuronas por capa.
   c) Función de inicialización.
   d) Función de activación.

   Después de declarar el modelo, se optimiza, se compila y por último se ajusta con los datos de validación y entrenamiento de todo el conjunto.

5 Resultados

Sin música, la vida sería un error.

Friedrich Nietzsche

En este Capítulo se analizará la batería de pruebas para poder comprobar qué algoritmos y parámetros son los más óptimos para el reconocimiento de instrumentos. En primer lugar se realizarán una serie de experimentos modificando los parámetros de cada uno de los algoritmos y por último se concluirá con una tabla resumen de los algoritmos que maximizan la probabilidad de acierto. Además, se realizará en cada uno de estos experimentos una set de pruebas tanto disminuyendo las dimensiones como utilizando la matriz original de datos.

5.1 Experimentos

5.1.1 Random Forest

El algoritmo de *Random Forest* tiene como parámetro modificable la profundidad que tienen los bosques de decisión.

Random Forest sin LDA o PCA

En la siguiente tabla se puede observar como cambia la probabilidad de acierto según la profundidad del bosque aleatorio.

Tabla 5.1 Tabla de experimentos con Random Forest.

Profundidad del bosque aleatorio	Probabilidad de acierto
5	70.74%
10	80.27%
15	79.59%
20	79.59%

Para poder ver el funcionamiento este algoritmo, el árbol de decisión elegido es el del caso de profundidad 5. (Se ha elegido este por temas de visibilidad).

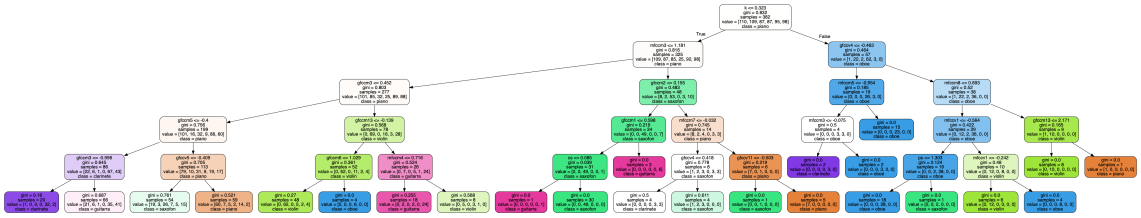

Figura 5.1 Árbol seleccionado por el algoritmo de *Random Forest* con una profundidad de 5.

A continuación se va a realizar un zoom para poder observarlo con más claridad. Este zoom estará dividido en cuatro bloques. Parte izquierda, parte centro izquierda, parte centro derecha y parte de la derecha:

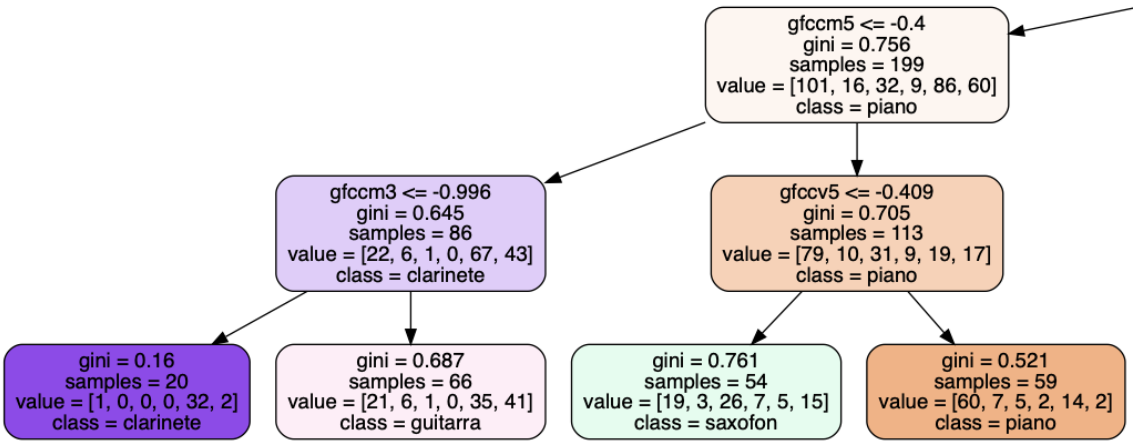

Figura 5.2 Bloque 1. Parte de la izquierda.

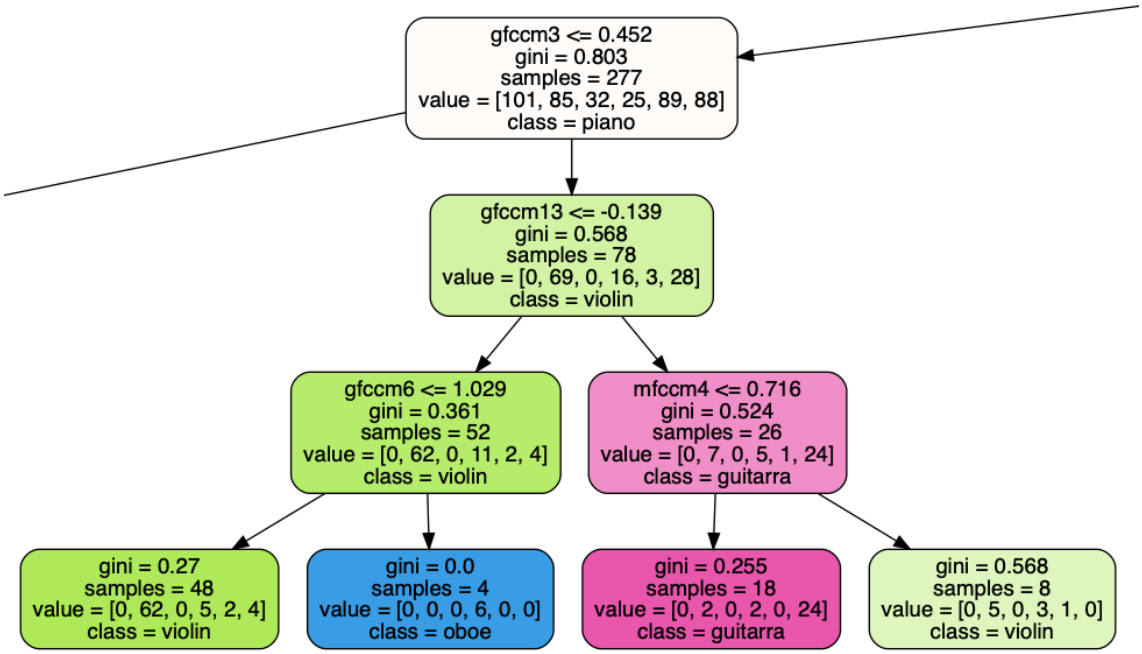

Figura 5.3 Bloque 2. Parte centro izquierda.

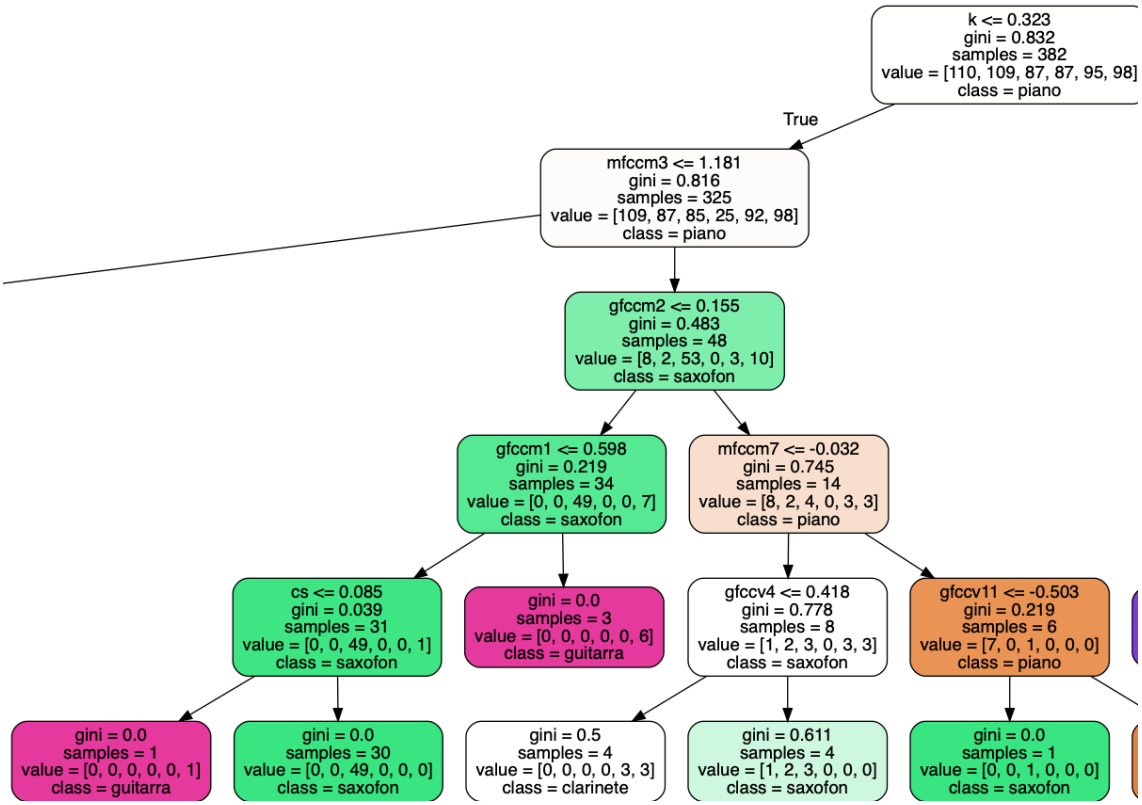

Figura 5.4 Bloque 3. Parte centro derecha.

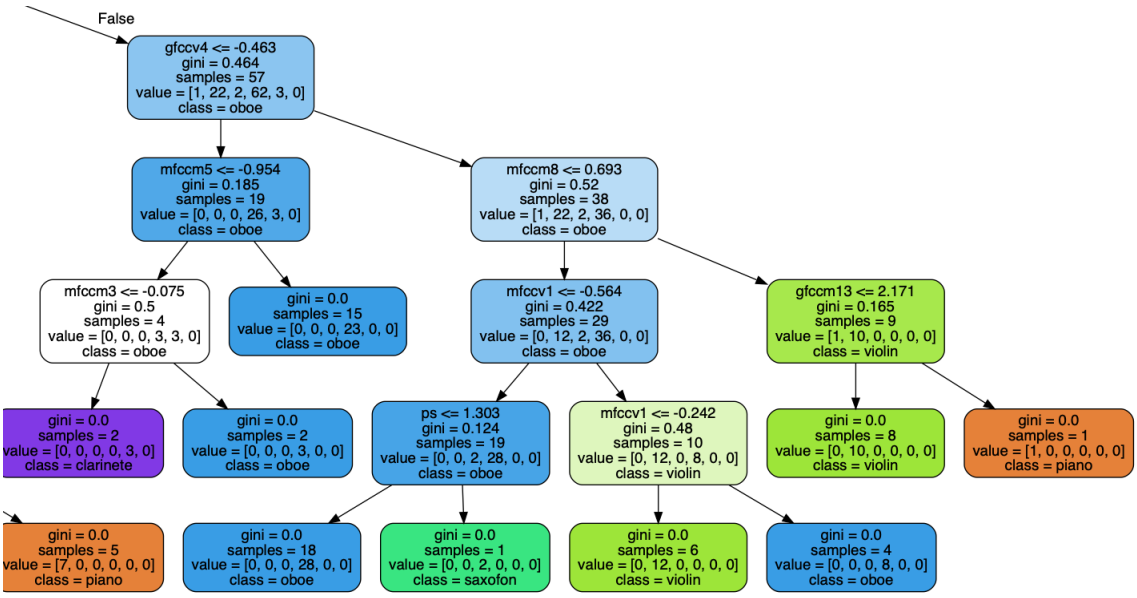

Figura 5.5 Bloque 4. Parte de la derecha.

Random Forest con LDA o PCA

En esta sección se va a añadir una batería de pruebas, primero minimizando la dimensión de la matriz de características y posteriormente usando el algoritmo de *Random Forest*.

Tabla 5.2 Tabla de experimentos con disminución de la matriz y a continuación utilizando el algoritmo de *random forest*.

LDA o PCA	Nueva dimensión	Profundidad del bosque aleatorio	Probabilidad de acierto
PCA	30	5	70.07%
PCA	40	5	73.47%
PCA	10	10	79.59%
PCA	20	10	73.47%
PCA	10	20	73.47%
PCA	30	20	71.42%
LDA	5	5	79.59%
LDA	5	10	80.27%
LDA	5	15	80.95%
LDA	5	20	80.95%
LDA	10	15	80.95%

Comentar que solo se han cogido las combinaciones que tienen más de un 70% de acierto. LDA es óptimo cuando tiene 5 componentes. Se ha realizado también algunas pruebas cambiando la dimensión de LDA pero se obtienen los mismos resultados que con LDA=5. Se puede observar algunas probabilidades de acierto se han visto mejoradas. Sin embargo, esta incremento de acierto, no es muy significativo para el caso de PCA. En cambio, cuando la profundidad del bosque es 5 y se utiliza LDA, la probabilidad de acierto aumenta casi en un 10%.
Para el caso LDA=5 y profundidad del bosque aleatorio = 15, se ilustrará a continuación la matriz de confusión.

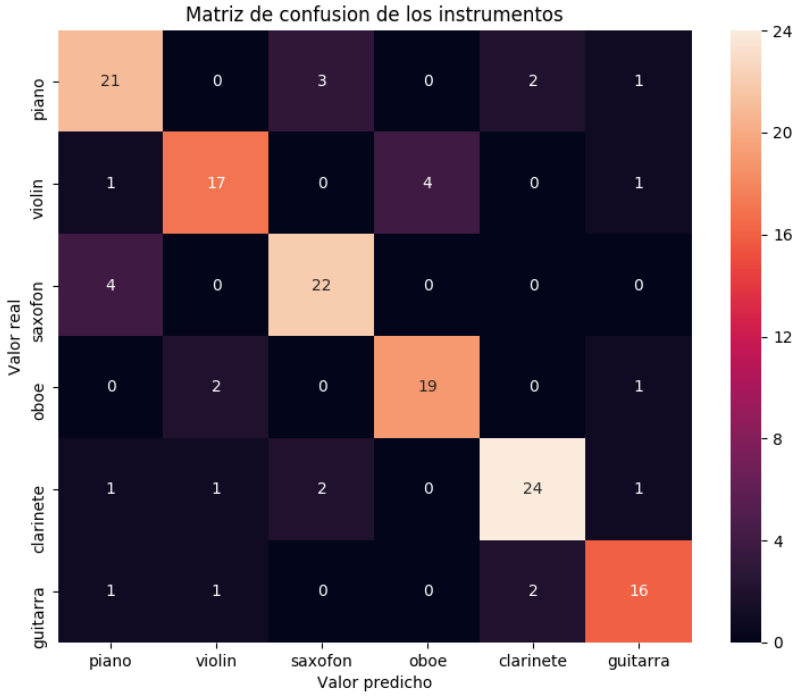

Figura 5.6 Matriz de confusión LDA=5 y profundidad de la random forest = 15.

A continuación se va a realizar un análisis de la matriz de confusión con el fin de definirla.
Una matriz de confusión es el resumen de resultados de las predicciones en una clasificación automática. Tanto en el eje X como en el eje Y se ponen los instrumentos, mientras que en el eje X están los valores

predichos de los instrumentos, el eje Y son los valores reales. La intersección de estos ejes, por ejemplo (PIANO, PIANO) son los valores acertados de muestras para las predicciones. Cada columna representará el número de predicciones para cada clase realizada por el modelo, y cada fila los valores reales por cada clase. Los valores que están en la diagonal son los valores verdaderos, es decir, las predicciones correctas que se han conseguido. Si por ejemplo estamos en el caso de mirando los ejes: (X,Y) (violin,piano) es de 1 es decir, para un audio de (validación-test) el clasificador se ha equivocado y ha puesto como si fuera ese audio un violin en vez de un piano.

5.1.2 KNN

Al igual que en el apartado anterior se va a realizar una conjunto de pruebas para el algoritmo de k-vecinos cercanos. Se va a realizar el mismo proceso. En primer lugar se verá el algoritmo sin minimizar la dimensión de la matriz de características y posteriormente se realizaran pruebas disminuyéndola.

KNN sin LDA o PCA

A continuación se expondrá una tabla de resultados modificando los k-vecinos más cercanos.

Tabla 5.3 Tabla de experimentos con KNN.

K-vecinos más cercanos	Probabilidad de acierto
1	87.07%
2	80.95%
5	75.51%
10	72.10%
15	67.34%

Se obtiene un valor óptimo con un solo vecino. Esto significa que la probabilidad máxima será el resultado de la mínima distancia euclídea entre puntos. Se puede observar la matriz de confusión de 1-vecinos cercanos.

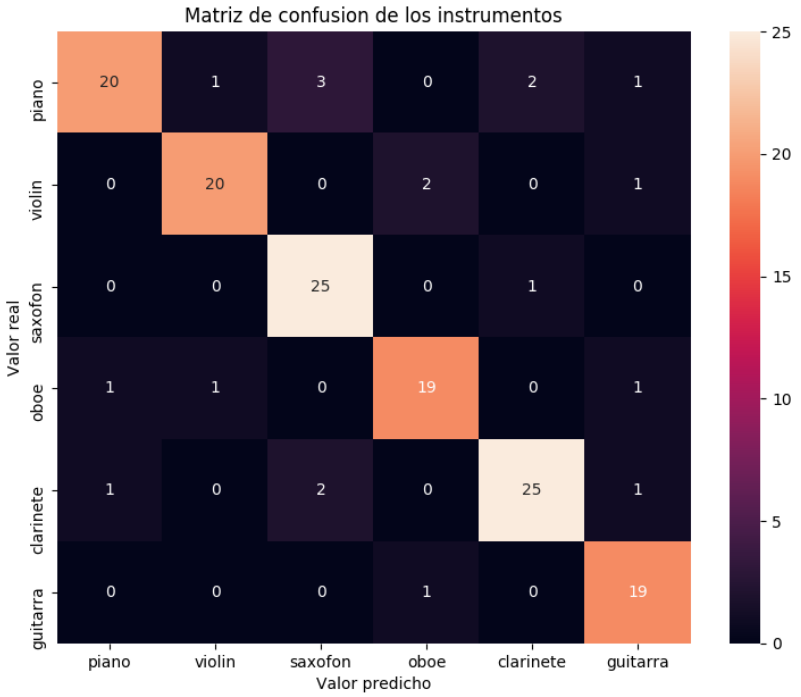

Figura 5.7 Matriz de confusión 1-vecinos cercanos.

KNN con LDA o PCA

Tabla 5.4 Tabla de experimentos con disminución de dimensionamiento de la matriz (PCA o LDA) y posteriormente clasificación con KNN.

LDA o PCA	Nueva dimensión	K-vecinos próximos	Probabilidad de acierto
PCA	30	1	82.99%
PCA	40	1	83.67%
PCA	50	1	86.39%
PCA	55	1	87.07%
PCA	55	2	80.95%
PCA	55	5	75.51%
LDA	5	1	80.27%
LDA	5	2	78.23%
LDA	5	5	84.35%
LDA	5	4	82.99%
LDA	5	10	83.67%

Se puede observar que para el valor óptimo de KNN (k = 1) no se ha obtenido ninguna mejora al introducir los algoritmos de minimización de la matriz de características. Además, al disminuir los coeficientes con PCA de 60 a 55 no existe ningún cambio entre ambos resultados, es decir, son iguales.
Sin embargo, para LDA aunque no se mejoren los resultados para k = 1, los demás valores sí se pueden ver optimizados.

5.1.3 Redes Neuronales

Como se ha comentado en (4.3.2) los parámetros modificables de las redes neuronales son más numerosos que los anteriores. Al existir número de capas, número de neuronas, diferentes funciones de inicialización y de activación, hace que la batería de pruebas sea muy interesante para poder observar el funcionamiento de este algoritmo. Se utilizará también el algoritmo PCA para disminuir la matriz de características.
Se va a dividir los experimentos de las redes neuronales en tres partes para poder seguir un orden ya que al haber tantos parámetros existen infinidad de combinaciones posibles:

1. Variación del número de capas.
2. Variación del número de neuronas.
3. Variación de las funciones de activación.

Y se van a fijar otros parámetros ya que al realizar este conjunto de pruebas, son con los que mejores resultados se obtienen:

1. Optimización con algoritmo *Adam*. Este algoritmo es una extensión del descenso de gradiente estocástico que recientemente ha visto una adopción más amplia para aplicaciones de deep learning. El método calcula las tasas de aprendizaje adaptativo individuales para diferentes parámetros a partir de estimaciones del primer y segundo momento de los gradientes. Calcula un promedio móvil exponencial del gradiente y el gradiente cuadrado, y los parámetros beta1 y beta2 controlan las tasas de decaimiento de estos promedios móviles [30].
2. Compilación con las pérdidas del *error cuadrático medio*.
3. Numero de *epochs = 150* y el *batch_size = 5*. Una *epoch o época* es cuando un conjunto de datos entero se pasa hacia adelante y hacia atrás a través de la red neuronal solo una vez. Además, no se puede meter todo el conjunto de datos directamente, por ello se dividen en bloques y se van metiendo las muestras de entrenamiento poco a poco. Estos bloques de datos se denominan *batch size*.
4. Función de inicialización. Siempre va a ser uniforme.

Redes Neuronales sin PCA

Al igual que los demás apartados, se va a proceder a realizar un conjunto de pruebas de redes neuronales sin disminuir la matriz de características. Se han realizado pruebas con ANN de dos capas y los resultados son

menores del 20% de acierto. Con lo cual se ha establecido un mínimo de tres capas.

Experimento 1: Modelo de red neuronal con tres capas.

Este modelo se tomará como referencia para los siguientes experimentos a realizar. Se obtiene un 67.35%. Un resultado fácilmente mejorable modificando algunos parámetros.

Tabla 5.5 Tabla de experimentos con redes neuronales de tres capas. Experimento 1.

Capas	Número de neuronas	Función de inicialización	Función de activación
Primera	12	Uniforme	ReLu
Segunda	40	Uniforme	Tangente hiperbólica
Tercera	6	Uniforme	Tangente hiperbólica

Experimento 2: Modelo de red neuronal con tres capas y variando el número de neuronas por capa y la función de activación

Este segundo modelo obtiene un 76.19% de probabilidad de acierto:

Tabla 5.6 Tabla de experimentos con redes neuronales de tres capas. Experimento 2.

Capas	Número de neuronas	Función de inicialización	Función de activación
Primera	20	Uniforme	ReLu
Segunda	55	Uniforme	ReLu
Tercera	6	Uniforme	Sigmoidea

Experimento 3: Modelo de una red neuronal de tres capas modificando el número de neuronas por capa

Este modelo ha obtenido un 75.51% de probabilidad de acierto. Sus parámetros son:

Tabla 5.7 Tabla de experimentos con redes neuronales de tres capas. Experimento 3.

Capas	Número de neuronas	Función de inicialización	Función de activación
Primera	12	Uniforme	ReLu
Segunda	40	Uniforme	ReLu
Tercera	6	Uniforme	Sigmoidea

Experimento 4: Modelo de una red neuronal de cuatro capas.

A continuación se ha añadido una capa a la red neuronal. Con esta modificación se ha obtenido un 81.63% de acierto en el reconocimiento automático.

Tabla 5.8 Tabla de experimentos con redes neuronales de cuatro capas. Experimento 4.

Capas	Número de neuronas	Función de inicialización	Función de activación
Primera	12	Uniforme	ReLu
Segunda	40	Uniforme	ReLu
Tercera	50	Uniforme	ReLu
Cuarta	6	Uniforme	Sigmoidea

Con el experimento cuatro se ha obtenido buenos resultados. Sin embargo, se van a observar una serie de gráficas que enseñan que este modelo se puede optimizar.

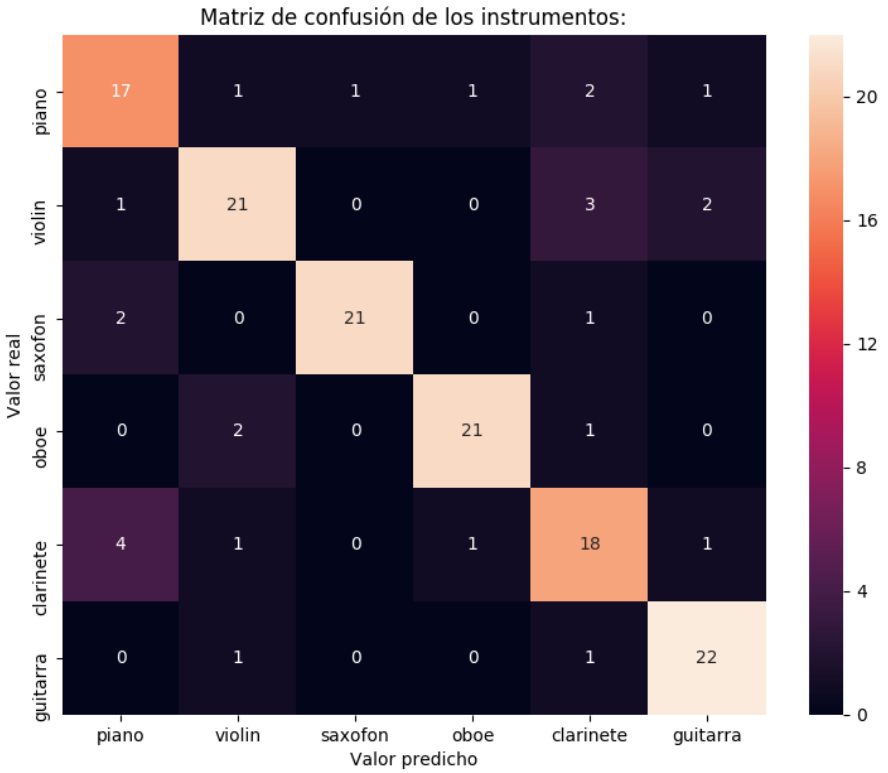

Figura 5.8 Matriz de confusión para una red neuronal de cuatro capas.

En la siguiente tabla se puede observar la precisión con la que se han clasificado cada uno de los instrumentos a reconocer:

Tabla 5.9 Tabla de precisión de clasificación de instrumentos por el experimento cuatro.

Instrumento	Precisión
Piano	71 %
Violin	81 %
Saxofón	95 %
Oboe	91 %
Clarinete	69 %
Guitarra	85 %

Además se puede observar que el clarinete es el instrumento con menos precisión para el reconocimiento automático. Puede ser que el piano, al ser eléctrico, posea un registro similar al clarinete. Además, en un principio, se escogieron los instrumentos clarinete-oboe ya que tenían timbres parecidos para que fuera más complicada la clasificación.
La gráfica 5.9 expresa el acierto en cada *epoch* tanto de las muestras de entrenamiento, como de las muestras de validación. Podemos decir que para cuatro capas se produce un efecto de Overfitting, es decir, el modelo está sobre entrenado, a medida que pasan las épocas, se va disminuyendo la probabilidad de acierto. En este caso, sería necesario reducir el número de epochs para obtener unos resultados más generales. Esto se desarrollará en el último experimento.

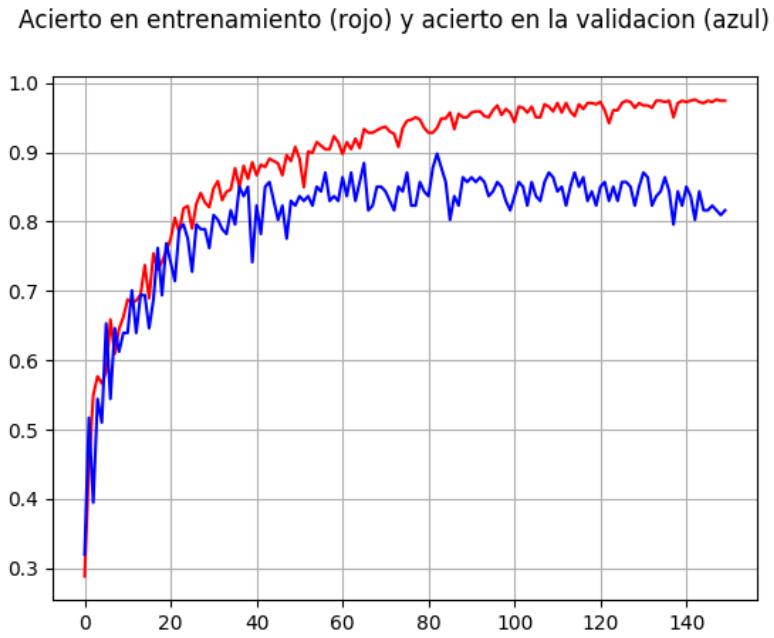

Figura 5.9 Gráfica de acierto de entrenamiento y de validación del experimento 4.

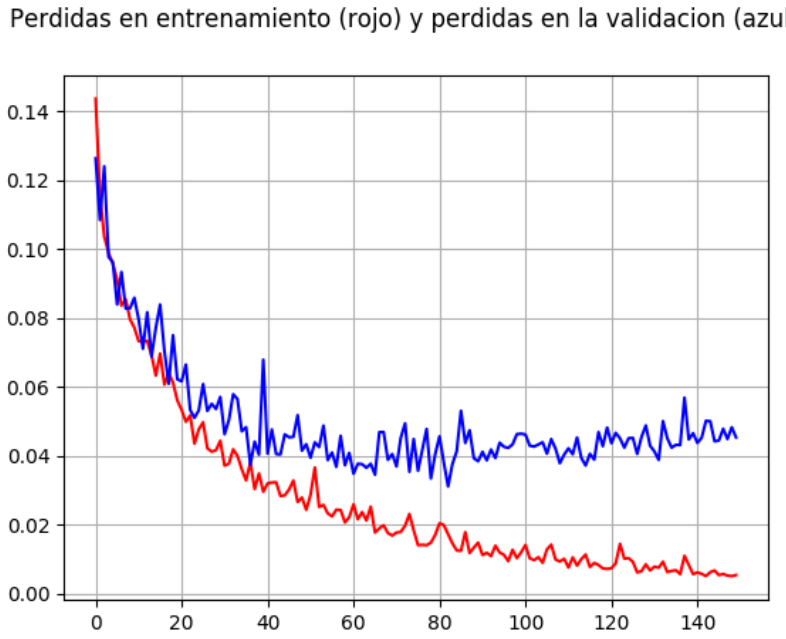

Figura 5.10 Gráfica de perdidas de entrenamiento y de validación del experimento 4.

Experimento 5: Modelo de una red neuronal de cuatro capas, 130 épocas y 3 de batch size.

Como se ha observado que se produce un efecto de *Overfitting* se va a realizar este último experimento para poder ver la diferencia entre un modelo que está sobre entrenado y un modelo cuyo entrenamiento no se ve afectado por el *Overfitting*. Los parámetros son exactamente iguales que en el experimento cuatro pero al tener 130 epochs y un tamaño de cada bloque de 3, se obtiene también un 81.63% de probabilidad de acierto. Sin embargo, es interesante poder comparar ambas gráficas. Mientras que en el experimento cuatro,

la tasa de acierto va disminuyendo a medida que se realizan más épocas, en el experimento 5 parece que va a empezar a desarrollar una cierta estabilidad.

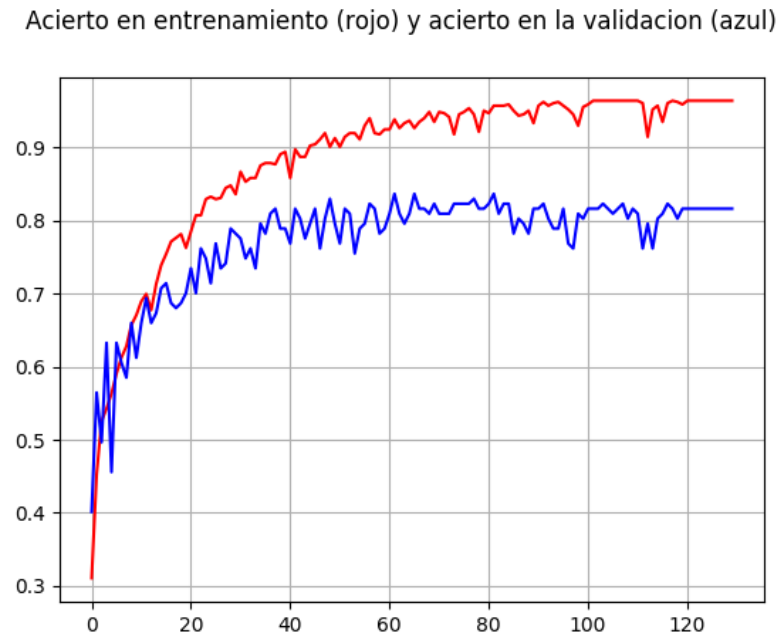

Figura 5.11 Gráfica de acierto de entrenamiento y de validación del experimento 5.

Figura 5.12 Gráfica de perdidas de entrenamiento y de validación del experimento 5.

Redes Neuronales con PCA

Esta es la última sección de la batería de pruebas. Se va a introducir una disminución de la matriz de características mediante el uso del algoritmo PCA. Se van a realizar algunas de las pruebas anteriores (las que

obtienen mejores probabilidades de acierto) variando las dimensiones.

Experimento 1: Modelo de red neuronal con tres capas y PCA

Los parámetros serán los mismos que los que se tienen en la tabla 5.6 (del segundo experimento) y se van a poner nuevas dimensiones para ver el funcionamiento de PCA.

Tabla 5.10 Tabla de minimización de la matriz de características con PCA y modelo de la tabla 5.6.

Dimensión PCA	Probabilidad de acierto
20	77.55%
30	80.95%
40	86.39%
55	83.67%

Cambiando la dimensión de la matriz a 40 con PCA obtenemos una clara optimización. Se ha mejorado con un 10% más de acierto.
Como es uno de los mejores resultados obtenidos se va a adjuntar su matriz de confusión y una tabla indicando la precisión de reconocimiento de cada instrumento.

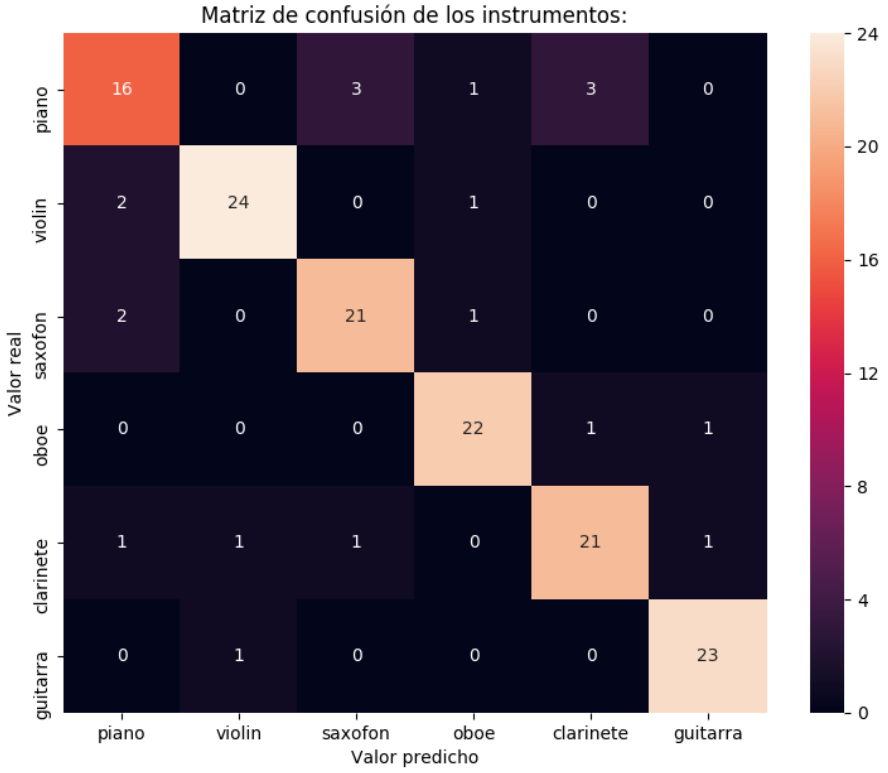

Figura 5.13 Matriz de confusión para dimensiones PCA = 40 y modelos de red neuronal de la tabla 5.6.

La tabla de precisión es la siguiente. Se puede seguir observando que los instrumentos que clasifica el algoritmo con más dificultad siguen siendo el piano y el clarinete.

Tabla 5.11 Tabla de precisión de clasificación de instrumentos por el experimento cuatro.

Instrumento	Precisión
Piano	76%
Violin	92%
Saxofón	84%
Oboe	88%
Clarinete	84%
Guitarra	86%

Experimento 2: Modelo de red neuronal con tres capas modificando las neuronas y PCA

Para este experimento se ha utilizado la tabla 5.7 del apartado anterior. Se realizará también un análisis

Tabla 5.12 Tabla de minimización de la matriz de características con PCA y modelo de la tabla 5.7.

Dimensión PCA	Probabilidad de acierto
20	77.55%
30	80.27%
40	83.67%
55	83.67%

Se puede observar que se han obtenido peores resultados que en el modelo anterior. Estas pruebas son experimentales, es por ello que se tiene que ir probando parámetros para analizar cuales son los más óptimos para nuestro algoritmo.

Experimento 3: Modelo de una red neuronal de cuatro capas y PCA.

Este es el último experimento que se va a realizar. Se va a utilizar el modelo de la tabla 5.8 del apartado anterior y como en los demás apartados, se va a hacer un set de pruebas modificando la matriz de características.

Tabla 5.13 Tabla de minimización de la matriz de características con PCA y modelo de la tabla 5.8.

Dimensión PCA	Probabilidad de acierto
20	77.55%
30	79.59%
40	82.31%
55	85.71%

5.2 Resultados óptimos

Por último se va a realizar una tabla comparativa de los mejores resultados de cada batería de pruebas.

Tabla 5.14 Tabla de resultados óptimos del proyecto.

Disminución de dimensiones	Algoritmo	Parámetros utilizados	Probabilidad de acierto
No	KNN	k=1	87.07%
PCA=55	KNN	k=1	86.39%
PCA=40	Red neuronal	Modelo tabla 5.6	86.39%
PCA=55	Red neuronal	Modelo tabla 5.8	85.71%
LDA=5	KNN	k=5	84.35%

Se han realizado bastantes más pruebas, pero sólo se han expuesto los resultados que superaban aproximadamente el 70% ya que, un resultado de menos de un 50% es poco significativo y carece de valor para este

proyecto.

6 Conclusiones y líneas futuras de investigación

En este proyecto se han desarrollado múltiples formas de reconocimiento de instrumentos. Todas estas pruebas pueden ser utilizadas en un futuro como base de investigaciones. Este capítulo va a exponer las conclusiones y las líneas futuras de investigación.

6.1 Conclusiones

Como primera conclusión, después de realizar toda la batería de pruebas, se puede afirmar que a veces con los algoritmos más sencillos se pueden obtener los mejores resultados. Es el claro ejemplo del KNN de un vecino. Tras un intenso análisis de redes neuronales, para este set de datos, parece ser que es más óptimo la clasificación según la distancia euclídea. Este resultado ha sido en sí, algo sorprendente.
También remarcar que se han realizado muchas más pruebas en la parte de resultados, pero no se han adjuntado debido a que su probabilidad de acierto era menor del 65% es el ejemplo de las redes neuronales de menos de tres capas, se obtiene un acierto de un 20% sin importar todas las modificaciones que se realicen (cambiar la función de activación, el número de neuronas, el optimizador etc).
Por otra parte, se ha podido observar que en algunas ocasiones, reducir la matriz de dimensiones no significa obtener mejores resultados. Pero también hay que tener en cuenta que en la mayoría de los casos, se ha podido observar una mejora.
Por último, los instrumentos que han causado mayor confusión han sido el piano eléctrico y el clarinete. Para la mayor parte de los algoritmos, estos dos instrumentos han sido los que más dificultades han aportado. Además, inicialmente, se pensaba que los instrumentos más similares iban a ser el clarinete y el oboe pues tienen un registro timbrico parecido. Sin embargo, el oboe ha sido bien reconocido en casi todas las pruebas, esto se puede observar claramente en las matrices de confusión aportadas en esta memoria.

6.2 Lineas futuras

Como se ha podido observar, existe una gran variedad de algoritmos de aprendizaje automático. Aunque en este proyecto se hayan analizado PCA, LDA, Random Forest, KNN y ANN, se pueden implementar otros, como es el ejemplo de máquina de soporte vectorial (SVM) o incluso redes neuronales convolucionales.
Teniendo la base de datos sólo haría falta estudiar los parámetros necesarios en los otros algoritmos e implementarlos. Esta parte es muy interesante ya que si por ejemplo no se hubiera probado el algoritmo KNN, nunca se habría sabido que es el óptimo para esta clasificación.
Por otra parte, el reconocimiento de instrumentos siempre puede servir de base para otras investigaciones futuras. La inteligencia artificial está en continuo desarrollo, y existen una gran variedad aplicaciones que podrían utilizar este proyecto con el fin de aumentar sus propiedades.

7 Anexo

7.1 Códigos realizados

7.1.1 Extracción de características

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import json
import csv

DIR_JSON   = '/Users/aurorasalgadodiazdelrio/Desktop/TFG/CARACTERISTICAS/JSON/120'
DIR_OUTPUT = '/Users/aurorasalgadodiazdelrio/Desktop/TFG/CARACTERISTICAS/CSV'

INSTRUMENTOS = ['piano', 'violin', 'saxofon', 'oboe', 'clarinete', 'guitarra']

features = []

for instrumento in INSTRUMENTOS:

    with open(DIR_JSON + '/' + instrumento + '.json', "r") as data_file:
        data = json.load(data_file) # esto es un diccionario

    # caracteristicas mfcc media de los coeficientes + varianza de los coeficientes +
    for audio in data.keys():
        caracteristicas_data = data[audio]['stats']['lowlevel']
        caracteristicas_aux = []

        #mfcc
        caracteristicas_aux.extend(caracteristicas_data['mfcc']['mean']) # mete la media de los instrumentos
        for varianza in range(0,13):
            caracteristicas_aux.append(caracteristicas_data['mfcc']['cov'][varianza][varianza]) # music extractor nos da la matriz de covarianza, como sabemos la varianza de cada uno de estos coeficientes del mfcc estan en la diagonal de esta.


```

```
30          caracteristicas_aux.extend(caracteristicas_data['gfcc']['mean']) #
                mete la media de los instrumentos
31          for var in range(0,13):
32              caracteristicas_aux.append(caracteristicas_data['gfcc']['cov'][
                    var][var])
33
34          # Centro de masa del espectro
35          caracteristicas_aux.append(caracteristicas_data['spectral_centroid
                ']['mean'])
36
37
38          # Energia espectral
39          caracteristicas_aux.append(caracteristicas_data['spectral_energy']['
                mean'])
40
41          #Prominencia del tono de un espectro
42          caracteristicas_aux.append(caracteristicas_data['pitch_salience']['
                mean'])
43
44          caracteristicas_aux.append(caracteristicas_data['spectral_rolloff
                ']['mean'])
45
46          caracteristicas_aux.append(caracteristicas_data['spectral_kurtosis
                ']['mean'])
47
48          caracteristicas_aux.append(caracteristicas_data['spectral_flux']['
                mean'])
49
50          caracteristicas_aux.append(caracteristicas_data['spectral_skewness
                ']['mean'])
51
52          caracteristicas_aux.append(caracteristicas_data['spectral_rms']['
                mean'])
53
54
55          #se agrega al final el tipo de instrumento que es
56          caracteristicas_aux.append(instrumento)
57
58          #se mete todo en una lista de listas para poder pasarlo a un archivo
                 .csv
59          features.append(caracteristicas_aux)
60
61  # se pasa del diccionario que teniamos en python a un archivo .csv para
        poder parsear despues los datos
62
63  with open(DIR_OUTPUT + '/' + 'features.csv', "w") as f:
64      writer = csv.writer(f)
65      writer.writerows(features)
```

7.1.2 Redes neuronales

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3
4  # En primer lugar se incluyen las librerias que vamos a utilizar
5  # Estamos usando Keras para hacer la clasificacion de los instrumentos y va
       a llamar por detras a TensowFlow
```

```
6
7  from keras.models import Sequential
8  from keras.layers import Dense
9  from keras import optimizers
10 from keras.utils import np_utils
11 import pandas
12 from sklearn.decomposition import PCA
13 import seaborn as sns
14 from sklearn.metrics import confusion_matrix
15 from sklearn.model_selection import train_test_split
16 from sklearn.preprocessing import LabelEncoder
17 import numpy
18 from matplotlib import pyplot as plt
19 from sklearn.metrics import accuracy_score
20 from sklearn.metrics import classification_report
21 # Se pone una semilla para realizar una segmentacion aleatoria de los datos
       y que se vaya repitiendo en cada ejecucion.
22 seed = 14
23 numpy.random.seed(seed)
24
25 # Se leen los datos y se parsean con la libreria pandas para hacerlo una
      estructura
26 datos = pandas.read_csv("features.csv", header=None)
27
28 # Se dividen los datos en dos subconjuntos, X sera todas las
      caracteristicas de los instrumentos e Y es la etiqueta de estos.
29 # Hay que tener en cuenta que en python, cuando se vayan a clasificar los
      datos solo se pueden tener numeros, hay que cambiar la Y
30 # para que en vez de instrumentos, sean numeros.
31 X = datos.values[:,0:60]
32 Y = datos.values[:,60]
33
34
35 # Se debe de pasar los instrumentos que son cadenas de caracteres a numeros.

36 encoder = LabelEncoder()
37 encoder.fit(Y)
38 Y_to_num = encoder.transform(Y)
39
40 # Se pasan los valores numericos anteriormente usados a codificacion de 1
      entre esos n.
41 Y_cod = np_utils.to_categorical(Y_to_num)
42
43 # Se hace una particion entre los datos que van a ser usados como
      entrenamiento, y los datos que van a usarse para validacion.
44
45 X_train, X_validation, Y_train, Y_validation = train_test_split(X,Y_cod,
      test_size = 0.2, random_state = seed, stratify=Y_cod)
46
47 # PCA
48
49 componentes = 55
50
51 pca = PCA(n_components=componentes, svd_solver = "full", whiten= False)
52 pca.fit(X_train)
53 X_train = pca.transform(X_train)
54 X_validation = pca.transform(X_validation)
```

```
55 X = pca.transform(X)
56
57
58 # Se crea el modelo directamente desde Keras. Como son 6 instrumentos los
       que van a ser clasificados, la ultima capa tiene que ser de 6.
59
60 model = Sequential()
61 model.add(Dense(12, input_dim=55, kernel_initializer = "uniform",
       activation="relu"))
62 model.add(Dense(40, kernel_initializer = "uniform", activation="relu"))
63 model.add(Dense(6, kernel_initializer = "uniform", activation="sigmoid"))
64
65 # Se va a utilizar el optimizador de Adam, es el que mejores resultados
       tiene.
66
67 adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, decay=0.0002)
68
69 # Se compila el modelo mean_squared_error
70
71 model.compile(loss="mean_squared_error", optimizer=adam, metrics=["accuracy
       "])
72
73 # Se ajusta el modelo, usando los datos de valiadacion anteriormente
       creados.
74
75 h = model.fit(X_train, Y_train, epochs=150, validation_data = (X_validation
       , Y_validation), batch_size=5, verbose=0)
76
77 # Se evaluan los resultados a partir de las muestras de validacion
78
79 scores= model.evaluate(X_validation,Y_validation)
80
81 # Resultados y porcentaje de aciertos
82
83 print ("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
84
85 # Graficas
86
87 plt.close("all")
88
89
90 acc = h.history["acc"]
91 loss = h.history["loss"]
92 acc_val = h.history["val_acc"]
93 loss_val = h.history["val_loss"]
94
95 f = plt.figure(1)
96 f.suptitle(u"Acierto en entrenamiento (rojo) y acierto en la validacion (
       azul)")
97 ax = f.add_subplot(111)
98 ax.plot(acc, "r")
99 ax.plot(acc_val, "b")
100 ax.grid(b=True)
101
102 f1 = plt.figure(2)
103 f1.suptitle(u"Perdidas en entrenamiento (rojo) y perdidas en la validacion
       (azul)")
```

```
ax1 = f1.add_subplot(111)
ax1.plot(loss, "r")
ax1.plot(loss_val, "b")
ax1.grid(b=True)


y_predict=model.predict(X_validation)
cm = confusion_matrix(Y_validation.argmax(axis=1), y_predict.argmax(axis=1)
    )
print cm


# Realizacion de la matriz de confusion:

cm_df = pandas.DataFrame(cm,
                  index = ['piano', 'violin', 'saxofon', 'oboe', '
                      clarinete', 'guitarra'],
                  columns = ['piano', 'violin', 'saxofon', 'oboe', '
                      clarinete', 'guitarra'])

plt.figure(figsize=(9,7))
sns.heatmap(cm_df, annot=True)
plt.title(u'Matriz de confusión de los instrumentos:')
plt.ylabel(u'Valor real')
plt.xlabel(u'Valor predicho')

print 'Accuracy Score :',accuracy_score(Y_validation.argmax(axis=1), y_
    predict.argmax(axis=1))
print 'Report : '
print classification_report(Y_validation.argmax(axis=1), y_predict.argmax(
    axis=1), target_names=['piano', 'violin', 'saxofon', 'oboe', 'clarinete
    ', 'guitarra'])

plt.show()
```

7.1.3 LDA y Random Forest

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-


import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report


datos = pd.read_csv("features.csv", header=None)
```

```
21
22
23
24 X = datos.iloc[:, 0:60].values
25 Y = datos.iloc[:, 60].values
26
27
28 X_train, X_validation, Y_train, Y_validation= train_test_split(X, Y, test_
       size=0.2, random_state=0)
29
30
31 sc = StandardScaler()
32 X_train = sc.fit_transform(X_train)
33 X_validation = sc.transform(X_validation)
34
35
36 lda = LDA(n_components=5)
37 X_train = lda.fit_transform(X_train, Y_train)
38 X_validation = lda.transform(X_validation)
39
40
41 classifier = RandomForestClassifier(max_depth=15, random_state=0)
42
43 classifier.fit(X_train, Y_train)
44 y_pred = classifier.predict(X_validation)
45
46 cm = confusion_matrix(Y_validation, y_pred)
47 print(cm)
48 cm_df = pd.DataFrame(cm,
49                      index = ['piano', 'violin', 'saxofon', 'oboe', '
                             clarinete', 'guitarra'],
50                      columns = ['piano', 'violin', 'saxofon', 'oboe', '
                             clarinete', 'guitarra'])
51
52 plt.figure(figsize=(9,7))
53 sns.heatmap(cm_df, annot=True)
54 plt.title('Matriz de confusion de los instrumentos:')
55 plt.ylabel('Valor real')
56 plt.xlabel('Valor predicho')
57 print('Accuracy: ' + str(accuracy_score(Y_validation, y_pred)))
58 plt.show()
59 print classification_report(Y_validation, y_pred, target_names=['piano', '
       violin', 'saxofon', 'oboe', 'clarinete', 'guitarra'])
```

7.1.4 LDA y KNN

```
1
2 #!/usr/bin/env python2
3 # -*- coding: utf-8 -*-
4
5
6 import numpy as np
7 import pandas as pd
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import train_test_split
11 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
12 from sklearn.svm import SVC
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn.metrics import confusion_matrix
15 from sklearn.metrics import accuracy_score
16 from matplotlib import pyplot as plt
17 import seaborn as sns
18 from sklearn.metrics import classification_report
19
20
21 datos = pd.read_csv("features_120_medias_10JUNIO.csv", header=None)
22
23
24
25 X = datos.iloc[:, 0:60].values
26 Y = datos.iloc[:, 60].values
27
28
29 X_train, X_validation, Y_train, Y_validation= train_test_split(X, Y, test_
       size=0.2, random_state=0)
30
31 # Se normalizan los datos
32
33 sc = StandardScaler()
34 X_train = sc.fit_transform(X_train)
35 X_validation = sc.transform(X_validation)
36
37 #LDA
38
39 lda = LDA(n_components=5)
40 X_train = lda.fit_transform(X_train, Y_train)
41 X_validation = lda.transform(X_validation)
42
43 classifier = KNeighborsClassifier(n_neighbors = 1)
44
45
46
47 classifier.fit(X_train, Y_train)
48 y_pred = classifier.predict(X_validation)
49
50 cm = confusion_matrix(Y_validation, y_pred)
51 print(cm)
52 cm_df = pd.DataFrame(cm,
53                      index = ['piano', 'violin', 'saxofon', 'oboe', '
                           clarinete', 'guitarra'],
54                      columns = ['piano', 'violin', 'saxofon', 'oboe', '
                           clarinete', 'guitarra'])
55
56 plt.figure(figsize=(9,7))
57 sns.heatmap(cm_df, annot=True)
58 plt.title('Matriz de confusion de los instrumentos:')
59 plt.ylabel('Valor real')
60 plt.xlabel('Valor predicho')
61 print('Accuracy: ' + str(accuracy_score(Y_validation, y_pred)))
62 plt.show()
63 print classification_report(Y_validation, y_pred, target_names=['piano', '
       violin', 'saxofon', 'oboe', 'clarinete', 'guitarra'])
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] *15th annual music information retrieval evaluation exchange*, https://www.music-ir.org/mirex/wiki/2019:Main_Page.

[2] *Biología - coordinación y locomoción*, https://sites.google.com/site/coordinacionylocomocion/home/2--componentes-del-sistema-nervioso.

[3] *Essentia. music extractor code*, https://github.com/MTG/essentia/blob/master/src/python/essentia/pytools/extractors/music_extractor.py.

[4] *Essentia. music extractor features*, https://essentia.upf.edu/documentation/streaming_extractor_music.html.

[5] *Essentia. open-source library and tools for audio and music analysis, description and synthesis*, https://essentia.upf.edu/documentation/.

[6] *Freesound. fsd: a dataset of everyday sounds*, https://annotator.freesound.org/fsd/.

[7] *Gammatone filter bank*, http://siggigue.github.io/pyfilterbank/gammatone.html.

[8] *Introducción al machine learning. k vecinos más cercanos (clasificación y regresión)*, http://www.pythondiario.com/2018/01/introduccion-al-machine-learning-9-k.html.

[9] *Keras*, https://keras.io.

[10] *Las 5vs que caracterizan el concepto de bigdata*, Noviembre, https://bigdata400.wordpress.com/2014/11/11/las-5-vs-que-caracterizan-el-concepto-de-big-data/.

[11] *Matplotlib 3.1.0*, https://matplotlib.org/.

[12] *Media cuadrática*, https://es.wikipedia.org/wiki/Media_cuadrática.

[13] *Music technology group*, https://www.upf.edu/web/mtg/about.

[14] *Numpy*, https://www.numpy.org/.

[15] *Pandas*, https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html.

[16] *Pitch salience*, https://essentia.upf.edu/documentation/reference/streaming_PitchSalience.html.

[17] *python-csv 0.0.11*, https://pypi.org/project/python-csv/.

[18] *Random forest*, https://es.wikipedia.org/wiki/Random_forest.

[19] *Recuperacóin de información musical*, https://es.wikipedia.org/wiki/Recuperación_de_información_musical.

[20] *Scikit-learn. machine learning en python*, https://scikit-learn.org/stable.

[21] *seaborn: statistical data visualization*, https://seaborn.pydata.org.

[22] *simplejson 3.16.0*, https://pypi.org/project/simplejson/ .

[23] *Spectral centroid*, https://en.wikipedia.org/wiki/Spectral_centroid.

[24] *Visual studio code*, https://code.visualstudio.com.

[25] Iñaki Inza y Pedro Larrañaga Abdelmalik Moujahid, *Clasificadores k-nn*, 08 2004.

[26] Hassan M. Ghaziri J. L. González Manuel Laguna Pablo Moscato Fan T. Seng Adenso Díaz, Fred Glover, *Optimización heurística y redes neuronales*, Editorial Paraninfo. S. A., 1996.

[27] Michele Alberti, Mathias Seuret, Vinaychandran Pondenkandath, Rolf Ingold, and Marcus Liwicki, *Historical document image segmentation with lda-initialized deep neural networks*, 2017.

[28] Basilio Sierra Araujo, *Aprendizaje automático: conceptos básicos*, PEARSON EDUCACIÓN, S.A., 2006.

[29] Carlos Baia, *Decision tree e random forest*, 2016, http://carlosbaia.com/2016/12/24/decision-tree-e-random-forest/ .

[30] Jason Brownlee, *Gentle introduction to the adam optimization algorithm for deep learning*, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ .

[31] Misha Denil, David Matheson, and Nando De Freitas, *Narrowing the gap: Random forests in theory and in practice*, 2014.

[32] J. Stephen Downie, *The music information retrieval evaluation exchange: A window into music information retrieval research*, vol. 29, 2008.

[33] Florian Eyben, *Real-time speech and music classification by large audio feature space extraction*, Springer, 2015.

[34] Rahana Fathima and Raseena P. E, *Gammatone cepstral coefficient for speaker identification*, vol. 2, 2013, https://pdfs.semanticscholar.org/64bb/5e102851386ac2bac526cbe1f24a7b8cdb6b.pdf .

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016.

[36] Mr. Nitin Goyal and Dr. R.K.Purwar, *Analyzing mel frequency cepstral coefficient for recognition of isolated english word using dtw matching*, vol. 3, https://pdfs.semanticscholar.org/15ce/b6976fbf7b8fd2d10fd0b86c825ba0ceeea3.pdf .

[37] Aleix M. Martínez and Avinash C. Kak, *Pca versus lda*, vol. 23, 01 2001.

[38] Yusnita mohd ali, *Mel filter banks basis functions using 20 mel-filters in the filter bank.*, https://www.researchgate.net/figure/Mel-filter-banks-basis-functions-using-20-Mel-filters-in-the-filter-bank_fig1_288632263.

[39] Erith Alexander Muñoz, *Modelo del perceptron de una capa (hsieh, 2009).*, https://www.researchgate.net/figure/Figura-29-Modelo-del-Perceptron-de-una-Capa-Hsieh-2009_fig4_280683488.

[40] Henri Paz, *Modelo del perceptron multicapa.*, https://www.researchgate.net/figure/Estructura-de-un-Perceptron-multicapa-Este-modelo-se-compone-de-la-siguiente-manera-o_fig2_281380920.

[41] Geoffroy Peeters, *A large set of audio features for sound description (similarity and classification) in the cuidado project*, vol. 1, 2004, http://recherche.ircam.fr/anasyn/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf .

[42] Alberto García Serrano, *Inteligencia artificial. fundamentos, práctica y aplicaciones*, 2ª ed., RC Libros, 2016.

[43] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien, *Linear discriminant analysis: A detailed tutorial*, vol. 30, 05 2017.

[44] Gonzalo Pajares Martinsanz y Matilde Santos Peña, *Inteligencia artificial e ingeniería del conocimiento*, Ed. RA-MA Editorial, 2005.

[45] José T.Palma Méndez y Roque Marín Morales, *Inteligencia artificial. técnicas, métodos y aplicaciones*, McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U., 2008.