Reconocimiento sonoro de instrumentos musicales

TITULACIÓN:

Máster Inteligencia Artificial

Curso académico:

2021-2022

Lugar de residencia, mes y año: Medellín, Colombia 30-05-2022

Alumno/a:

Rodriguez Villamizar, William Steve

D.N.I:

1098685961

Director:

Escrig Pérez, Helio

Convocatoria:

Segunda

Orientación:

Virtual

Créditos:

12 ECTS

viu Universidad Internacional de Valencia

Agradecimientos

En primer lugar, quiero agradecer a Dios y seguido mi familia, a mi esposa e hijos quienes con su amor me llenan de complacencia infinita y motivación sin limites.

A todos mis profesores, por la paciencia que ha tenido conmigo, por aguantar todas las dudas y preguntas infinitamente largas.

William Steve Rodríguez Villamizar

Colombia, 2022

Resumen

Con el presente trabajo se pretende abordar la clasificación de instrumentos musicales en una señal de audio y a su vez razonar cuales son las características de audios que son mas razonables para lograr la clasificación. En este caso clasificar: Piano, Violín, Viola, Violonchelo, Clarinete, Fagot, Bocina, Oboe, Flauta, Clave y Contrabajo, para ello se han dividido los audios en audios de 2 segundos y extraído 26 características a cada audio, a posterior se ha aplicado PCA al 92% y se han utilizado algoritmos de IA para llevar a cabo la clasificación y finalmente se verá la tabla de resultados y el modelo seleccionado.

Abstract

The present work is intended to approach the classification of musical instruments in an audio signal and at the same time to reason which audio characteristics are more reasonable to achieve the category. In this case, classified: Piano, Violin, Viola, Cello, Clarinet, Bassoon, Horn, Oboe, Flute, Harpsichord, and Contrabass, for this the audios have been divided into 2-second audios and 26 characteristics have been extracted from each audio, subsequently, PCA has been applied to 92% and AI algorithms have been used to carry out the classification and finally, the results table and the selected model will be seen.

Tabla de Contenido

Indice de tablas

Indice de imágenes

1. Introducción

El presente trabajo tiene como fin tomar un conjunto de datos de composiciones musicales del género clásico y poder clasificar los diferentes instrumentos que intervienen durante la canción, para ello se aplicará la metodología CRISP-DM como base atacar este tipo de problemas de inteligencia artificial, al final se entrenan varios modelos de IA y se comparan los resultados obtenidos.

1.1. Objetivos

La finalidad de este proyecto es entrenar un modelo de inteligencia que pueda definir que instrumentos musicales están sonando en un audio, se pretende reconocer 11 instrumentos, estos son:

- Piano
- Violín
- Viola
- Violonchelo
- Clarinete
- Fagot
- Bocina
- Oboe
- flauta
- Clave
- Contrabajo

Para ello se usara un dataset de kaggle, una web que contiene confiables y amplios datasets para diversos proyectos de ciencia de datos, big data y machine learning, en especifico se usara la base de datos de (musinet, 2020) [1].

A su vez se procederá a aplicar sobre los audios de entrada del modelo una extracción de características, según lo propone (Essentia, 2021)[2], en esto solo se elegirán unas características a usar, a fortuna (Sanket Doshi, 2018)[3], nos

define las características mas relevantes y que en su trabajo dieron importantes resultados, y al igual que El, se usará la librería (librosa, 2015)[4], la cual es una librería para extraer características de audios.

Luego de la extracción de características se creara un archivo .csv en el cual se incorporan estas características mencionadas, archivo que se usara para el modelo.

Finalmente se evaluara el modelo y se definirá la confianza, precisión y exactitud del mismo.

2. Estado del arte

Durante una exhaustiva investigación de trabajos relacionados se ha revisado entre otros el trabajo de Aurora Salgado5, donde en su trabajo usa un dataset que contiene audios de instrumentos específicos (violín, piano, etc), cada audio tiene solo un instrumento, pero su proceso de tratamiento de audios es un punto de partida crucial que se nos permite aprovechar su experiencia para nuestro tratamiento de datos.

Valorando el trabajo del Museo Chileno de Arte Precolombino [25], donde antes de poder clasificar los instrumentos musicales realizan una extracción de características de forma similar a como lo hace Aurora y con otras librerías, pero dado fuerza a que este es el camino a seguir.

Seguidamente se estudio el trabajo realizado por la universidad de Washington6, donde recopilaron y clasificaron un amplio dataset (musinet1) y entrenaron varios modelos usando scratch7 y redes neuronales convolucionales, funciones de activación ReLu, maxpooling, entre otras consideraciones logran resultados mas que admirables, pero en particular bastante llamativo como aplicar el PCA a conjunto de datos de tipo audio, PCA que nos viene bien y que usaremos mas adelante dado que en este trabajo demostró resultado no menos que excelentes.

En particular revisando también el trabajo de JUAN SEBASTIÁN MÉNDEZ HERNÁNDEZ8, donde con las características de audio de MFCC, demuestran que esta característica es fundamental para la clasificación de instrumentos, para el cual le prestaremos especial atención a la extracción de esta característica, la cual se explicara a mas detenimiento en capitulo posterior.

También y profundizando en la forma como MANUEL ENRIQUE ALDANA SÁNCHEZ [26] ataca el problema de clasificación de partituras abre un camino de como al tener cierta atención sobre cuando usar o no usar el PCA en audios para reducir sus características, lo que se aplicará como base de experiencia al momento de configurar el PCA.

Sin olvidar mencionar a Silvia Jiménez Gómez[27] donde consigue crear un modelo de red neuronal capaz de generar audio, si bien, este trabajo no trata de generar audio, las capas de las neuronas, funciones de activacion que ella

expone son de vital importancia cuando en este trabajo diseñemos el modelo RNA.

Finalmente usaremos lo aprendido por Juan Sebastian Gómez Cañón9, que le en experiencia luego de crear su clasificador de instrumentos musicales usando redes neuronales, confirma el porque a mayor cantidad de datos se obtienen mejores métricas positivas en el entrenamiento y test de modelos de IA, en especial en dataset de audios, experiencia que nos viene bien pues solo tenemos 330 audios (muestras), mientras que en todas las menciones anteriores se contaban con mas de 5000 muestras, por lo cual mas adelante usaremos una técnica para aumentar la cantidad de muestras.

Cabe resaltar que en temas de IA, el tratamiento de datos no ha sido tan profundizado en temas de audio como lo ha sido vasta-mente investigado para temas de visión por computadora (por poner un ejemplo), sin embargo se ataca el problema teniendo como base el estado del arte y se propone unos ajustes que no aparecen en la documentación original.

3. 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

3.1. Que es IA?

La Inteligencia Artificial (IA) es la combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano [15]. Una tecnología que todavía nos resulta lejana de una maquina pensante, pero muy real en ejecución de tareas generalmente luego de un entrenamiento previo.

3.2. Que es Machine learning

Para definir el machine learning prefiero usar la definición de IBM[16] ya es muy completa “Machine learning es una forma de la IA que permite a un sistema aprender de los datos en lugar de aprender mediante la programación explícita. Sin embargo, machine learning no es un proceso sencillo. Conforme el algoritmo ingiere datos de entrenamiento, es posible producir modelos más precisos basados en datos. Un modelo de machine learning es la salida de información que se genera cuando entrena su algoritmo de machine learning con datos. Después del entrenamiento, al proporcionar un modelo con una entrada, se le dará una salida. Por ejemplo, un algoritmo predictivo creará un modelo predictivo. A continuación, cuando proporcione el modelo predictivo con datos, recibirá un pronóstico basado en los datos que entrenaron al modelo.”

3.3. Que es Deep learning

Igualmente IBM[16] define muy bien lo que es deep learning así: “El deep learning es un método específico de machine learning que incorpora las redes neuronales en capas sucesivas para aprender de los datos de manera iterativa. El deep learning es especialmente útil cuando se trata de aprender patrones de datos no estructurados. Las redes neuronales complejas de deep learning están diseñadas para emular cómo funciona el cerebro humano, así que las computadoras pueden ser entrenadas para lidiar con abstracciones y problemas mal definidos. Las redes neuronales y el deep learning se utilizan a menudo en el reconocimiento de imágenes, voz y aplicaciones de visión de computadora.”, esto es mas practico verlo a travez de una imagen:

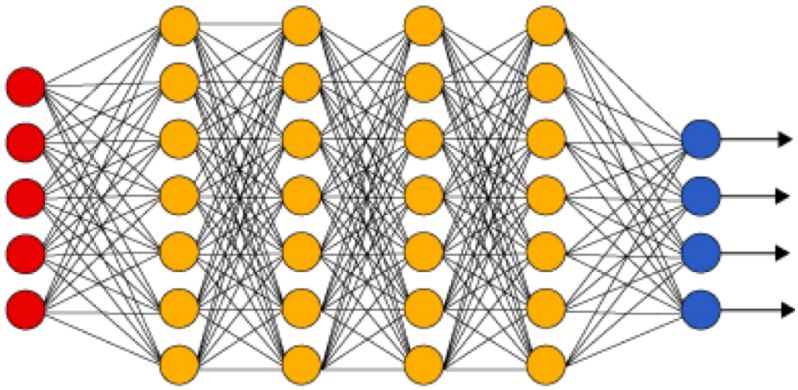


Imagen 1: Ejemplo red neuronal

Acá se puede observar como están interconectadas las neuronas y como cambian información de estas, la idea es que cada neurona aprenda una parte del conocimiento y al sumar todo el conocimiento, la red neuronal podrá atacar el problema en cuestion.

3.4. Que es aprendizaje supervisado

Siendo IBM[16] gran precursor de avances en inteligencia artificial, nos define el aprendizaje supervisado como: "El aprendizaje supervisado comienza típicamente con un conjunto establecido de datos y una cierta comprensión de cómo se clasifican estos datos. El aprendizaje supervisado tiene la intención de encontrar patrones en datos que se pueden aplicar a un proceso de analítica. Estos datos tienen características etiquetadas que definen el significado de los datos. Por ejemplo, se puede crear una aplicación de machine learning con base en imágenes y descripciones escritas que distinga entre millones de animales."

3.5. Que es PCA

La web ciencia de datos [17] estipula el PCA como "un método estadístico que permite simplificar la complejidad de espacios muéstrales con muchas dimensiones a la vez que conserva su información. Pertenece a la familia de técnicas conocida como unsupervised learning, El método de PCA permite por lo tanto "condensar" la información aportada por múltiples variables en solo unas pocas componentes. Esto lo convierte en un método muy útil de aplicar" en algoritmos de inteligencia artificial.

3.6. Que es RandomForest

"RandomForest (Bosque Aleatorio), es una técnica de aprendizaje automático", "es un conjunto (*ensemble*) de árboles de decisión combinados con bagging. Al usar bagging, lo que en realidad está pasando, es que distintos árboles ven distintas porciones de los datos. Ningún árbol ve todos los datos de entrenamiento. Esto hace que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, unos errores se compensan con otros y tenemos una predicción que generaliza mejor." una imagen representativa seria la siguiente:

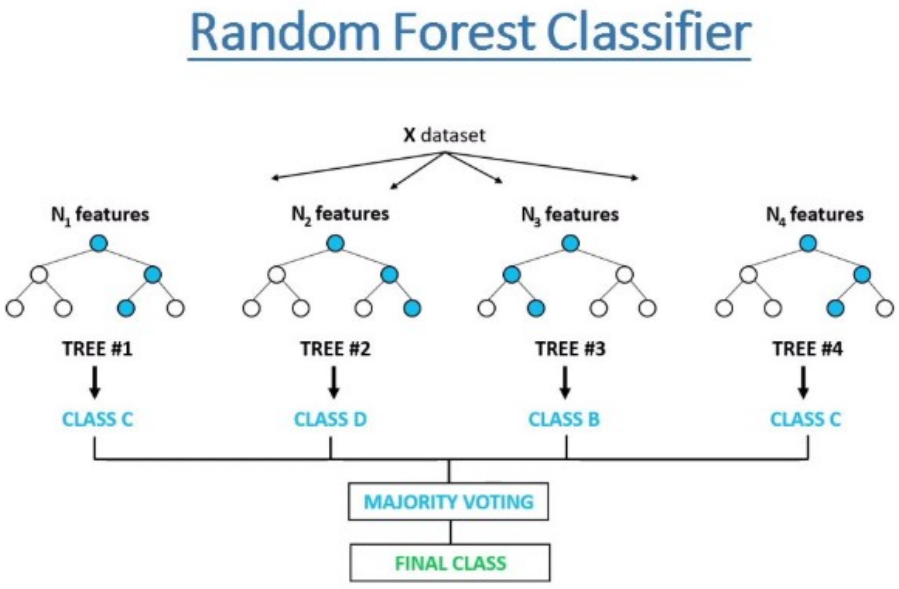


Imagen 2: Ejemplo de RandomForest, tomado

3.7. Que es LogisticRegression

"La Regresión Logística Simple, desarrollada por David Cox en 1958, es un método de regresión que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa. Una de las principales aplicaciones de la regresión logística es la de clasificación binaria, en el que las observaciones se clasifican en un grupo u otro dependiendo del valor que tome la variable empleada como predictor." [19], en este algoritmo la predicción se aproxima al binario mas cercano, una imagen que lo podría representar seria:

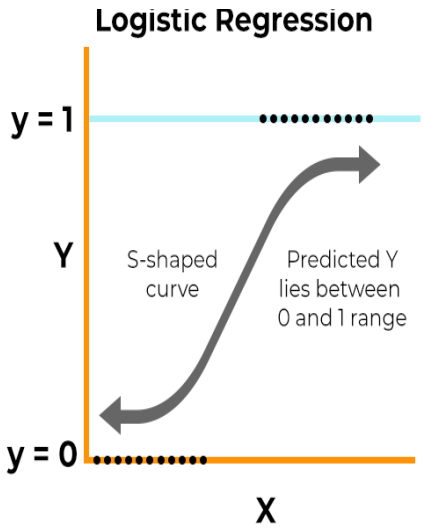


Imagen 3: Ejemplo LogisticRegression, tomado

3.8. Que es DecisionTreeClassifier

IBM[19] nos brinda una definición muy completa que vale la pena rescatar: "Un árbol de decisión es un algoritmo de aprendizaje supervisado no paramétrico, que se utiliza tanto para tareas de clasificación como de regresión. Tiene una estructura de árbol jerárquica, que consta de un nodo raíz, ramas, nodos internos y nodos hoja."

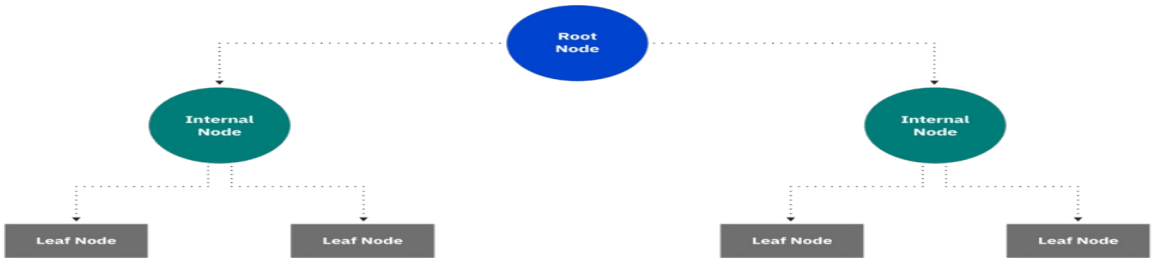


Imagen 4: Tomado de IBM [19]

"Este tipo de estructura de diagrama de flujo también crea una representación fácil de digerir de la toma de decisiones, El aprendizaje del árbol de decisiones emplea una estrategia de divide y vencerás mediante la realización de una

búsqueda codiciosa para identificar los puntos de división óptimos dentro de un árbol. Este proceso de división se repite de forma recursiva de arriba hacia abajo hasta que todos o la mayoría de los registros se hayan clasificado bajo etiquetas de clase específicas" [19]

3.9. Que es DummyClassifier

Un gran precurso de conceptos y ejercicios de inteligencia artificial es towardsdatascience[21] quien define a "un modelo clasificador que hace predicciones sin tratar de encontrar patrones en los datos. El modelo predeterminado esencialmente analiza qué etiqueta es más frecuente en el conjunto de datos de entrenamiento y hace predicciones basadas en esa etiqueta. Pero, antes de seguir adelante y construir un clasificador ficticio, necesitamos saber cómo comparar el modelo en cuestión con el clasificador ficticio. ", para representarlo simplemente se usa una matriz de confusión:

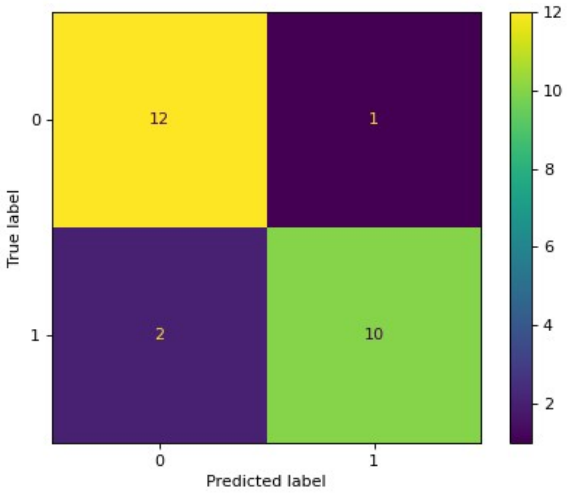


Imagen 5: Matriz de confusión

3.10. Que es KneighborsClassifier

Otra web que suelo consultar mucho es aprendemachinelearning quien define el"**K-Nearest-Neighbor** es un algoritmo basado en instancia de tipo supervisado de Machine Learning. Sirve esencialmente para clasificar valores buscando los puntos de datos "más similares" (por cercanía) aprendidos en la etapa de entrenamiento y haciendo conjeturas de nuevos puntos basado en esa

clasificación.”, esto simplemente es calcular a cual clase es mas cercano, por ejemplo:

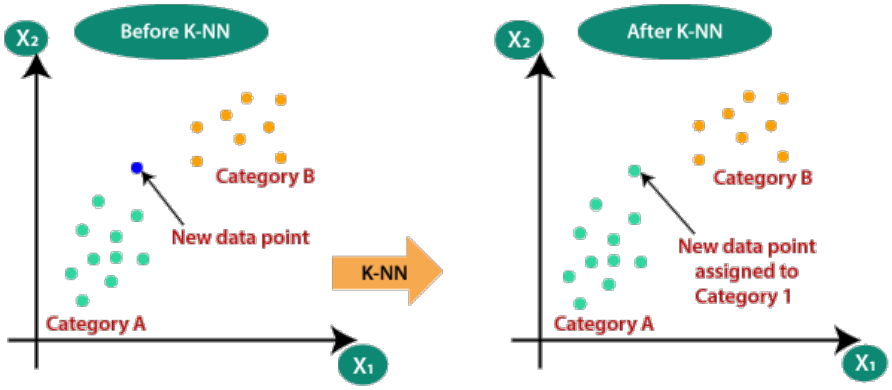


Imagen 6: Ejemplo KneighborsClassifier, tomado

3.11. Que es GaussianNB

Una web que encontré recientemente es programmerclick[23], quien tiene un apartado bastante amplio y claro sobre el algoritmo de Naive Bayes, a groso modo: “Entre todos los algoritmos de clasificación de aprendizaje automático, Naive Bayes es diferente de la mayoría de los otros algoritmos de clasificación. ”, a diferencia de los algoritmos anteriormente mencionados los cuales son discriminativos “,es decir, aprenden directamente la relación entre la salida de características Y y la característica X, o la función de decisión Y = f (X), o distribución condicional P (Y | X). Pero Naive Bayes es un método de generación, es decir, encuentra directamente la distribución conjunta P (X, Y) de la salida de la característica Y y la característica X, y luego usa P (Y | X) = P (X, Y) / P (X) inferido. Naive Bayes es muy intuitivo y no requiere muchos cálculos, tiene amplias aplicaciones en muchos campos.”[23], un ejemplo sería:

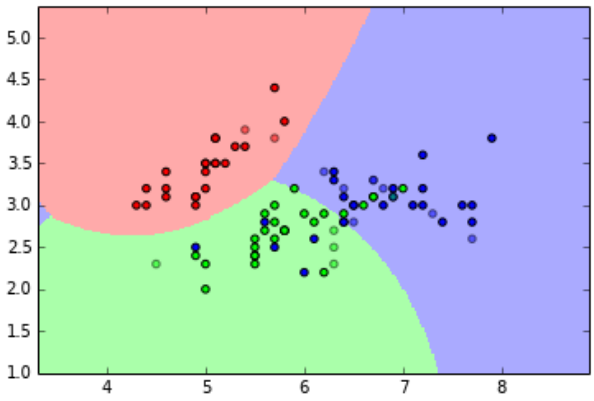


Imagen 7: Ejemplo clasificación

4. Dataset

Para poder llevar a cabo este trabajo se ha hecho una búsqueda de un dataset que estuviera etiquetado, pero sobre todo que fuera confiable en su fuente para poder usarlo como medio de este trabajo, en esta búsqueda varios trabajos, entre ellos, como el realizado por John Thickstun, Zaid Harchaoui y Sham Kakade6 donde usaron el dataset de musinet y Scratch para enseñar a un modelo a aprender diferentes características de audios, que si bien en este trabajo no se usara scratch o se usara el mismo enfoque, fue suficiente para entender que este es dataset adecuado para este trabajo.

En esto hay una comunidad dedicada a reunir diferentes datasets para diferentes usos de inteligencia artificial (IA para abreviar), en esto se hace referencia a kaggle que es una web altamente enriquecida con variedad de paquetes de datos para modelados de diferentes algoritmos.

En kaggle se ha logrado encontrar un conjunto de datos que es preciso al proporcionar un dataset apropiado para este trabajo, este es llamado como el conjunto de datos de musinet1.

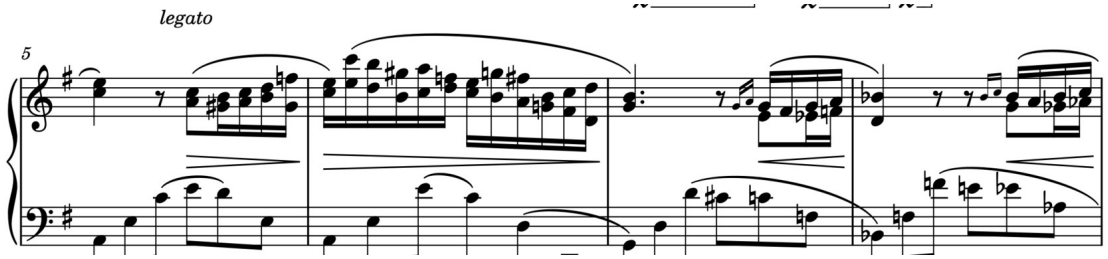


Imagen 8: Imagen representativa dataset

El cual está constituido por 330 grabaciones de música clásica con licencia libre, junto con más de 1 millón de etiquetas anotadas que indican el tiempo preciso de cada nota en cada grabación, el instrumento que toca cada nota y la posición de la nota en la estructura métrica de la composición. Las etiquetas se adquieren a partir de partituras musicales alineadas con grabaciones mediantes deformación dinámica del tiempo. Las etiquetas son verificadas por músicos calificados, donde se estima una tasa de error de etiquetado del 4%, a continuación un resumen del dataset:

Compositor	**cantidad de audios**	**tiempo total (segundos)**
Beethoven	157	65149
Bach	67	11041
Schubert	30	15188
Mozart	24	9386
Brahms	24	11531
Cambini	9	2577
Dvorak	8	3343
Ravel	4	1643
Faure	4	1963
Haydn	3	888
TOTAL	**330**	**122709**

Tabla 1: Resumen dataset

Con esto se puede observar que como valor adicional a la descripción oficial del dataset tenemos 122.709 segundos de grabación, esta es una nota importante pues jugaremos con este dato en el momento de preparar los datos, esto debido a que hasta el momento solo se tienen 330 registros de diferentes autores, que si bien es un numero de composiciones que en si, es un valor importante, para entrenar nuestros futuros modelos se va a incrementar la cantidad de registros con una técnica propuesta en este trabajo para que el modelo pueda generalizar mejor el aprendizaje de los datos.

Cabe resaltar que este paquete de datos es un conjunto de audios en .wav con un peso en disco (luego de descomprimir) de 33,5GB.

También se distinguen los instrumentos con sus id:

1. Piano
2. Violín
3. Viola
4. Violonchelo
5. Clarinete
6. Fagot
7. Bocina
8. Oboe
9. Flauta
10. Clave
11. Contrabajo

Este dataset fue diseñado para atacar varios problemas:

- Identificar las *notas* interpretadas en momentos específicos de una grabación.
- Clasificar los *instrumentos* que intervienen en una grabación.
- Clasificar al *compositor* de una grabación.
- Identifique *inicio* de las notas en una grabación.
- Prediga la *siguiente nota* en una grabación, condicionada por la historia.

Entre ellos este trabajo atacará: Clasificar los *instrumentos* que intervienen en una grabación.

5. Metodología CRISP-DM

Antes de empezar a atacar los objetivos de este trabajo vamos a entender el proceso de cómo se va a buscar la solución al problema. Para ello se usará la metodología CRISP-DM, el cual es una forma perfecta para atacar un problema de ciencia de datos para modelados de algoritmos de IA, lo cual viene perfecto para este problema.

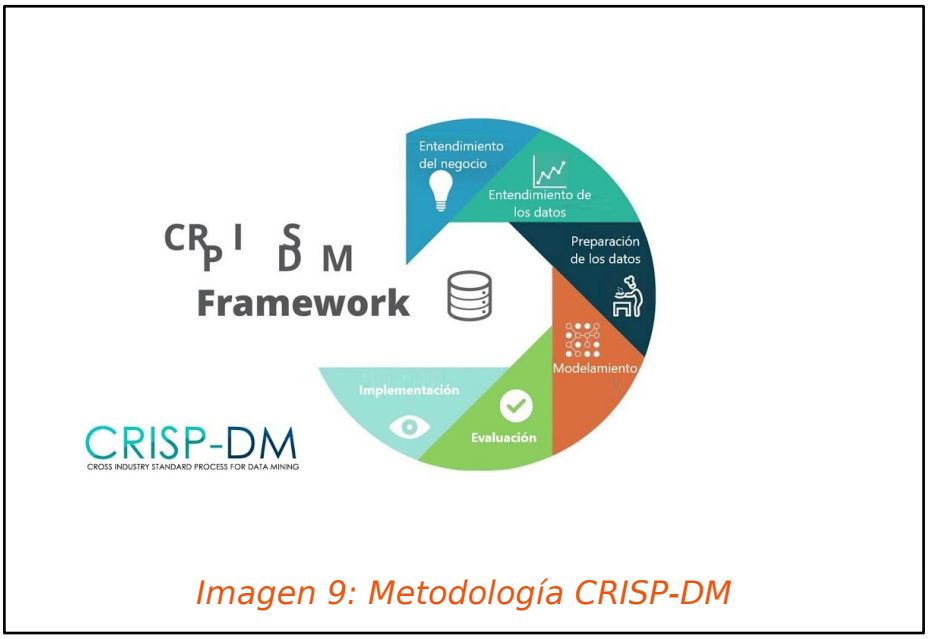


Imagen 9: Metodología CRISP-DM

El **entendimiento del negocio** hace referencia a entender los objetivos, entender el problema y tener claro la meta que se deberá alcanzar al final de la implementación, para este trabajo, como se ha mencionado anteriormente se quiere entregar un audio al modelo y que este en respuesta indique que instrumentos musicales han sonado en ese audio.

El **entendimiento de los datos** es un análisis del dataset, revisar cada dato, revisar las etiquetas, ver que datos aparecen con ruido o alguna anomalía que se pueda identificar, etc.

La **preparación de los datos** nos conlleva a trabajar sobre los datos para solucionar lo detectado en el punto anterior, separar datos, hacer incremento de la información con alguna técnica de data-incrementation, normalizar o estandarizar la información, extraer las características de los datos, quizás, hacer reducción de dimensionalidad, extraer los componentes principales, quitar

datos null, ausentes y por supuesto convertir a vectores, matrices y tensores los datos (paso crucial pues los modelos trabajan con matrices o tensores).

El **modelado**, es quizás la parte mas llamativa del proceso, acá es donde modificamos el algoritmo, dejamos fluir la creatividad para tratar de crear un modelo de inteligencia artificial que pueda aprender de los datos, se entrena ese modelo, se valida y se pone a prueba las habilidades del programador al momento de diseñar el algoritmo.

La **evaluación**, es usar del dataset el conjunto de datos de test, el cual es una pequeña información aislada del resto para poner a prueba el modelo diseñado y entrenado en el paso anterior, con el fin de entregar al modelo datos nuevos, frescos, desconocidos y ver qué métricas entrega, y así evaluar la eficiencia del mismo, esta evaluación define si el modelo está listo o no para pasar a producción.

Implementación es precisamente ese paso a producción cuando el modelo ha demostrado que es capaz de procesar efectivamente nuevos datos y se pone en un entorno de producción, generalmente a través de una API, ya sea en AWS o en un servidor local.

6. Preparación de los datos

6.1. Metodología a aplicar

En este apartado se definirá el proceso a seguir para preparar los datos, para ello y considerando la experiencia de Aurora Salgado [5], donde nos muestra que para tratar una muestra de audio la forma recomendada es extraer las características del audio y crear un vector donde cada posición del mismo es una característica de esta extracción, lo único es que Aurora extrae casi todas las características de las que son posible extraer (>35), por lo que me decidí en comparar, para ver cuales son las mas relevantes, con el trabajo de Sanket Doshi [3] quien da mucha valiosa importancia al croma, rms, centroide espectral, ancho de banda, reducción espectral, el cruce por cero y el MFCC y el trabajo de Juan Sebastian Mendez [8], fortalece algunos de ellos y enfoca un valor que resalta la importancia del MFCC.

Si bien Juan Sebastian usa mayormente el MFCC, Sanket Doshi y Aurora tiene un punto en común con las características mencionadas, con ello tenemos 6 características a extraer y 20 que entrega el MFCC, para un total de 26 características.

Seguido a esto y aplicando lo aprendido por el trabajo de la universidad de Washington [6] a esas 26 posiciones del vector resultante le aplicaremos el PCA para reducir aún mas la dimensionalidad de este vector.

6.2. Split para datos de X segundos

Otro punto a considerar viene relacionado con lo expuesto por Juan Sebastian Gómez Cañón [9], donde estipula que la cantidad de datos que se usa en el entrenamiento del modelo es un factor crucial para el aprendizaje del mismo, por ello se propone realizar un incremento de la cantidad de los datos.

Lastimosamente en audios es algo mas difícil que en imágenes donde haciendo corrimientos, ajustes de color, giro de imagen, entre otras, el data incrementation es fácil de realizar, mientras que en audios no es tan fácil, por ello una idea fue tomar los audios y separarlos en partes iguales, de esta manera, un audio se podría convertir en X audios (con sus respectivas etiquetas) y de esta manera se estaría haciendo un incremento de datos.

Ya se ha expuesto que se tienen 330 muestras de sonido, pero esa cantidad de registros es muy pobre para entrenar los algoritmos de aprendizaje supervisado, por ello se realizará un incremento de datos mediante un particionado de los datos, es decir, se tomara una canción X y la dividiremos en fragmentos de n segundos.

A su vez para cada fragmento de audio se buscara en el correspondiente banco de etiquetas, la(s) etiqueta(s) que corresponda para cada trozo del audio original, por ejemplo, si dividimos en secciones de 2 segundos un audio de 100 segundos se tendrán 50 muestras de audio.

Haciendo esto también se reduce el tamaño del vector que representa el audio, y para cada uno de esos 50 muestras se tendrán 11 salidas etiquetadas (binarias), para 11 instrumentos, una por cada etiqueta posible según el dataset, cabe aclarar que para algunas de estas salidas simplemente será cero, es decir, este instrumento no suena en este audio, esto se hace para poder estandarizar la cantidad de salidas igual para todas las muestras.

Al aplicar este proceso a todos los audios se tendrá un incremento de muestras de forma significativa, esto aplica tanto para los datos train como para los datos de test, así:

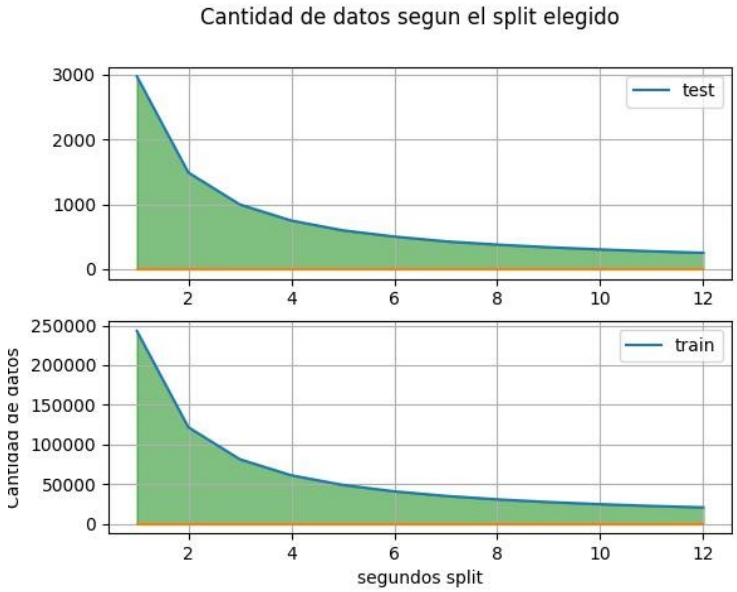


Imagen 10: Cantidad datos según split tiempo a los audios

Esta división también afecta el tamaño del vector que representa cada audio (antes de la extracción de características que se aplicará mas adelante), una relación se vería con el siguiente grafico:

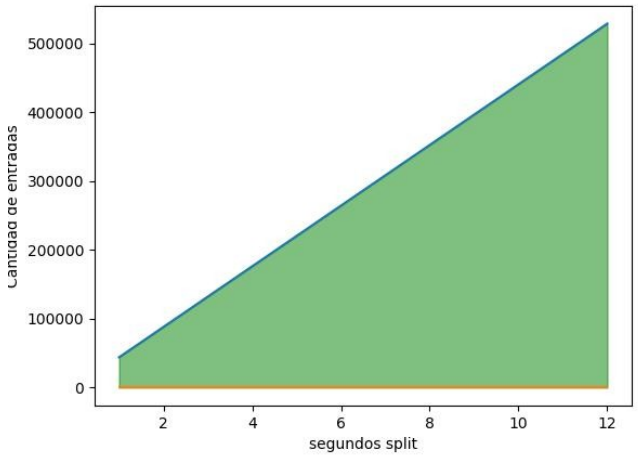


Imagen 11: Tamaño vector entrada según split tiempo a audios

En esto se logra intuir que entre mayor sea el split, mayor tamaño tendrá el vector de entrada.

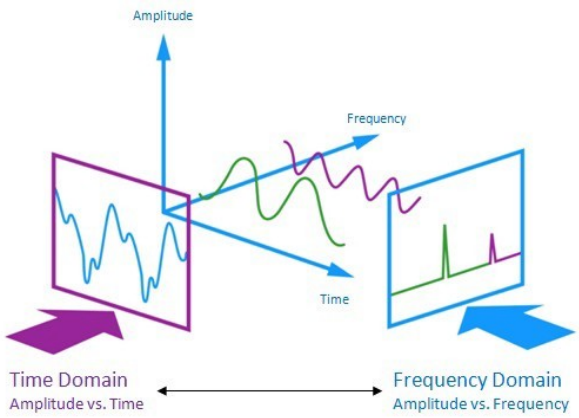
En ello se puede ver que al hacer el split pequeño, no solo se aumentan las muestras, sino que también se reduce el tamaño del vector, ahora solo falta definir cual es el valor del split de tiempo que sea mas apropiado para el entrenamiento de los modelos.

En este punto de este trabajo aun no se definirá el valor del split a usar, esto se hace en el apartado de modelado, por lo que por ahora centraremos la atención en extraer las características de cada fragmento de audio, en esto hay varios métodos para extraer las características de audios, los cuales se expondrán en el siguiente apartado.

7. Extracción características y PCA

La extracción de características es una parte muy importante para analizar y encontrar relaciones entre diferentes cosas. Los datos proporcionados de audio no pueden ser entendidos por los modelos directamente para convertirlos en un formato comprensible. Se utiliza la extracción de características. Es un proceso que explica la mayor parte de los datos de forma comprensible. La extracción de características es necesaria para los algoritmos de clasificación, predicción y recomendación.

En la literatura de tratamiento de señales, un audio es una señal tridimensional en la que tres ejes representan el tiempo, la amplitud y la frecuencia.



En los siguientes puntos de este capitulo se va a profundizar en diferentes características que se pueden extraer a un audio.

Se usara Librosa, una librería de python para analizar y extraer características de una señal de audio

7.1. Taza cruce por cero

Una señal de audio es una variación donde la señal pasa varias veces entre valores positivos y negativos, estos cambios es lo que define cada tonalidad e intensidad del sonido, esta es la tasa de cambio de signo a lo largo de una señal, es decir cambia de positiva a negativa y viceversa, esto es muy usado para el reconocimiento de voz, a gran percusión en valor es mayor, como es el caso del metal o rock.

Por ejemplo, para el siguiente audio se puede identificar que hay tres cruces por cero:

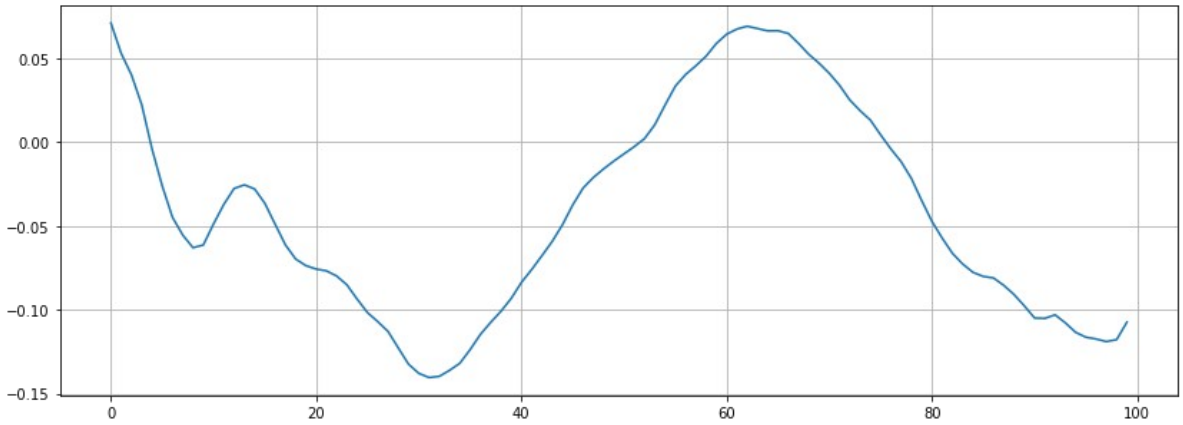


Imagen 12: Fragmento de audio ampliado para identificar los cruces por cero, tomado de 3

7.2. Centroide espectral

Este inicia done se encuentra el "centro de masa"[1] de un sonido, y se calcula como la media ponderada de las frecuencias presentes en el sonido. Si no hay variación de frecuencias entonces el centroide espectral estaría alrededor de un centro. 10

7.3. Reducción espectral

Es la frecuencia por debajo de la cual se encuentra un porcentaje especifico de energía espectral.

7.4. RMS

Es el valor de la raíz cuadrada media (RMS) para cada cuadro, ya sea de las muestras de audio y/o de un espectrograma, que se emplea para conocer la media aproximada de potencia 11.

7.5. croma

Se define el espectro de un sonido como la representación de la distribución de energía sonora de dicho sonido en función de la frecuencia. El espectro es

1

importante porque la percepción auditiva del sonido es de naturaleza predominantemente espectral.12

7.6. ancho de banda

El ancho de banda espectral es la longitud de la extensión de frecuencias, medida en hercios (Hz), en la que se concentra la mayor potencia de la señal13.

7.7. MFCC — Coeficientes centrales de frecuencia Mel

Esta característica es uno de los métodos más importantes para extraer una característica de una señal de audio y se usa principalmente cuando se trabaja con señales de audio. Los coeficientes centrales de frecuencia de MEL (MFCC) de una señal son un pequeño conjunto de características (generalmente entre 10 y 20) que describen de manera concisa la forma general de una envolvente espectral14.

7.8. Resumen extracción

Con esto ya se ha extraído las características de la señal musical.

Para resumir: para cada audio se tendrá el siguiente vector de respuesta:

croma	rms	centroide espectral	ancho banda	reducción espectral	cruces por cero	MFCC
1 valor	1 valor	1 valor	1 valor	1 valor	1 valor	20 valores

Tabla 2: Vector de salida luego de extraer las características del audio

Es decir para cada audio, sin importar el tamaño del vector de entrada, se convertirá luego de la extracción de características en un vector de tamaño 26 valores.

Como el tamaño del vector de entrada no define el tamaño del vector de salida (resultado de la extracción de características), entonces ya el tamaño del vector no es un factor que tenga preocupación para entrenar el modelo, lo único aún por resolver es la cantidad de datos, tema que se solucionará en el modelado.

7.9. PCA

Luego de extraer las características a todos los audios se tendrán 26 características, pero para fines prácticos, aun es un vector algo amplio, por lo que se realizó sobre los datos un análisis de cada una de esas características, también llamado componentes, dando el siguiente resultado:

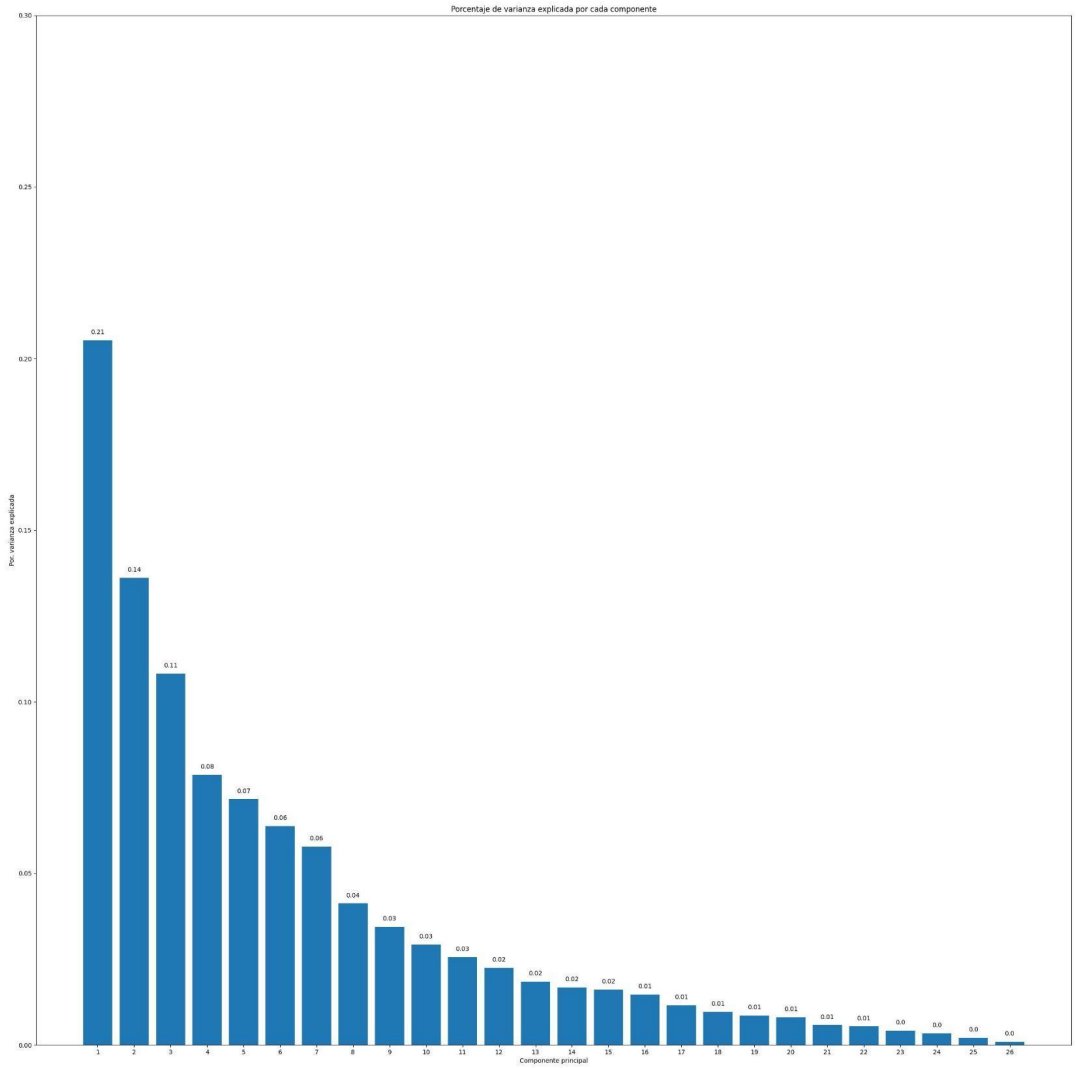


Imagen 13: Cantidad de información que contiene cada componente

Como se puede observar, las ultimas componentes contienen muy poco de la información original, así que con el PCA, evaluemos que valor seria mas

apropiado como nueva cantidad de componentes, en pocas palabras el vector 26 posiciones lo vamos a reducir a un valor menor, por ejemplo 21 posiciones, de esta manera al modelo que se entrene le resultará mas fácil aprender la generalidad de los datos y dará mejores métricas positivas en la evaluación del modelo, para ello se debe sacrificar un poco de la información, buscando que sea muy poco o mantener al menos un 90% o mas de la información, pero cual seria el porcentaje de la información que se mantendría luego de una reducción de dimensionalidad, para facilitar esta decisión se ha construido el siguiente grafico:

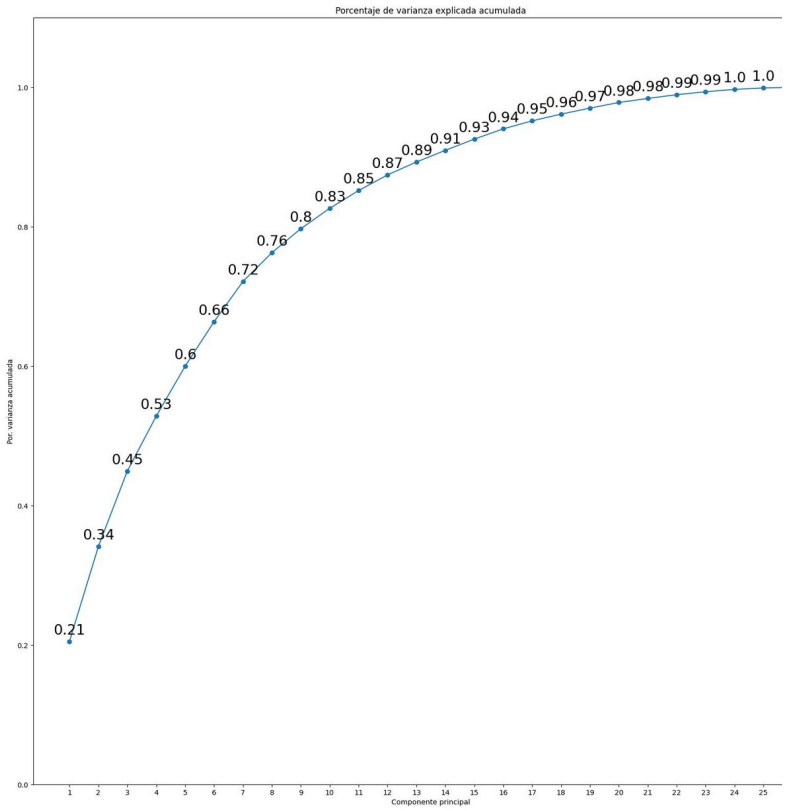


Imagen 14: Porcentaje de la información para las nuevas componentes

Con este análisis de resultados de reducción de dimensionalidad, se puede observar que al mantener las 15 primeras características se puede conservar cerca del 93% del total de la información, 15 nuevas componentes diferentes a las componentes originales, dicho de otra manera: al reducir 42% el tamaño del vector de 26 posiciones (tamaño original=26, tamaño luego de PCA=15), puedo

mantener el 93% de la información, es decir, sacrificando un aproximado al 7% de la información puedo reducir la cantidad de entradas a casi la mitad, lo que es un buen negocio, y que va a beneficiar exponencialmente al modelo a entrenar.

En resumen, se tendría lo siguiente:

split segundos	vector entrada	cantidad datos		extraccion de caracteristicas	PCA (%)														
		train	test		85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
1	44100	243106	2968	26	12	12	12	13	13	14	15	15	16	17	17	18	20	21	23
2	88200	121700	1488	26	11	12	12	13	13	14	14	15	16	16	17	18	19	21	23
3	132300	81248	996	26	11	12	12	13	13	14	14	15	16	16	17	18	19	21	23
4	176400	61014	748	26	11	12	12	13	13	14	14	15	16	16	17	18	19	21	23
5	220500	48878	598	26	11	12	12	12	13	14	14	15	16	16	17	18	19	21	23
6	264600	40780	502	26	11	11	12	12	13	14	14	15	16	16	17	18	19	21	23
7	308700	34996	428	26	11	11	12	12	13	14	14	15	16	16	17	18	19	21	23
8	352800	30666	378	26	11	11	12	12	13	13	14	15	15	16	17	18	19	21	23
9	396900	27290	338	26	11	11	12	12	13	13	14	15	15	16	17	18	19	21	23
10	441000	24604	304	26	11	11	12	12	13	13	14	15	15	16	17	18	19	21	23
11	485100	22388	276	26	11	11	12	12	13	13	14	15	15	16	17	18	19	21	23
12	529200	20550	252	26	11	11	12	12	13	13	14	14	15	16	17	18	19	21	23

magen 15: Comparación de nuevas componentes vs porcentaje de información

En resumen, se usará un PCA que me entregue solo 15 componentes, de esta manera mantendremos poco mas del 92% de la información.

8. Definición configuraciones de datos para modelo

8.1. Elección del split en segundos a usar

Ahora ha llegado el momento de tomar la decisión crucial que va a definir cual es el split a usar, para ello no se uso una decisión aleatoria o una de alguna bibliografía debido a que esta división para incrementar los datos surgió durante conversación con el tutor, por lo que se uso el método de ensayo y error, anteriormente en la imagen 7, se mostró la posibilidad de usar de 1 a 12 como valor de segundos para el split, para ello se muestra el siguiente gráfico que resume un entrenamiento para cada split:

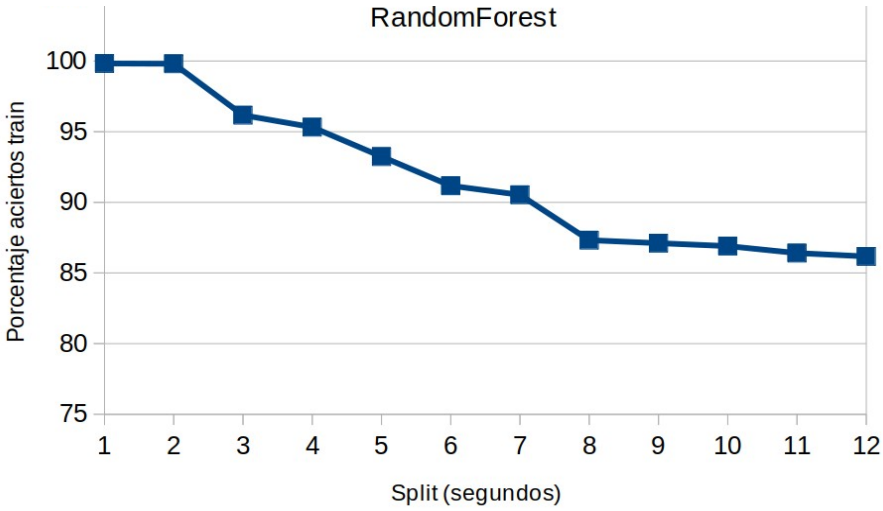


Imagen 16: Porcentaje Aciertos luego de train según el split en el algoritmo RandomForest (algoritmo elegido solo para generar el grafico)

El anterior grafico es el resultado de varios entrenamientos, para ello, para cada audio se ha hecho cada split expuesto y cada vector resultante se aplicó la extracción de características y PCA (vector de 15 posiciones resultante para cada audio), luego se creó un archivo csv que contenía en las filas cada vector de cada audio, con esto, se realizo un train usando el algoritmo RandomForest (con hiperparametros por default) con cada archivo de cada split y con los

resultados de cada entrenamiento se genero el grafico anterior, esto para determinar cual split era el mas adecuado.

Se puede observar que hay dos puntos de inflexión, el primero es el split a 7 segundos donde se obtiene un porcentaje de aciertos de 90,53% y otro punto de inflexión en el split a 2 segundos donde se logra un porcentaje de aciertos de 98,6%.

Teniendo 330 audios iniciales en un total de 243400 segundos, se ha decidido en partir cada audio en mini-audios de 2 segundos, donde teníamos un porcentaje de aciertos superior al 95% (por ello se eligen los dos segundos) y cantidad necesaria de datos para lograr un porcentaje mayor, con esto, las 330 muestras se convierten en 121700 muestras, lo que le da un valor de conveniencia para lograr en el entrenamiento del modelo una generalidad de los datos bastante alta.

Para puntualizar, se usa el split de dos segundos pues según la imagen 8, es que mejor nos promete resultados prometedores en el entrenamiento.

8.2. Función de evaluación a usar

Hasta ahora no se ha hablado de accuracy, Recall u otros, esto debido a que sciki-learn para las salidas multiclase no dispone de unas métricas que se puedan usar con facilidad, ni siquiera la matriz de confusión (para multiclase), por ello se creo una función que contabiliza la cantidad de aciertos (cuando la predicción es clase X y la salida etiquetada es también clase X) y los no aciertos como falsos, entonces se tiene una balanza donde en un extremo están los aciertos y en el otro las equivocaciones, función que devuelve el valor en el lado de los aciertos en forma de porcentaje.

8.3. División de datos en train, valid, test

Cada dato sera un vector de 88200 valores, que seguido de la extracción de características se tendrán 26 valores y finalmente luego del PCA se tendrá 15 valores, es decir, para el entrenamiento, se tendrá una matriz de 15 columnas y 121700 muestras, con esto ahora ya se puede pasar a entrenar el modelo de inteligencia artificial.

Kaggle en sus datasets siempre entrega dividido el set de train y el set de test, pero a mi gusto, siempre que trabajo con kaggle me gusta tener mi propio set de test para que el test dado por kaggle sean mis datos que validen mi pre-

producción, en esto mi set de train le realizo un split de datos dejando el 80% como mínimo para el train el 20% (generalmente) para validación y el set de datos que kaggle destina para test, usarlo para el test, siendo así se tiene (con split a 2 segundos):

- Train: 102160 muestras
- Valid: 24340 muestras
- Test: 744 muestras (dadas por kaggle)

9. Entrenamiento modelo

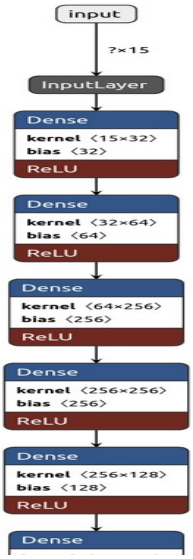
Los autores citados han usado el deep learning para el desarrollo de este trabajo, así que este fue el camino inicial al usar, para ello partiendo del archivo CSV constituido una matriz de 15X102160 se ha diseñado dos modelos de red neuronal, usando la librería de keras con backend de tensorflow 2.1, que par fin de diferenciarlos los he titulado como: básico y experimental.

Adicionalmente y dado que los autores encontrados en el estado del arte se inclinaron directamente por las redes neuronales mayormente, tenía la pregunta de el porque habían tomado esa decisión por ello quise por iniciativa propia en este trabajo comparar con algunos modelos de inteligencia artificial clásicos y en base a esta comparación determinar el porque de la decisión de estos autores, en ello se decidió entrenar algunos modelos vistos durante el Máster, estos son: RandomForestClassifier, LogisticRegression, DecisionTreeClassifier, DummyClassifier, KneighborsClassifier y GaussianNB, y en base a esto comparar los resultados, para estos modelos se usó el MultiOutputClassifier, GridSearchCV y 10-Kfold para entrenar varios modelos del mismo tipo de forma simultanea (con ligeros cambios) y quedarnos con el mejor.

Finalmente se presenta una tabla resumen comparando los resultados y se elegirá el modelo con las mejores métricas en la comparación.

9.1. Modelo RNA basico

Para el primer modelo de red neuronal se ha entrenado un modelo muy sencillo de 8 capas, las capas ocultas usaran la función de activación ReLu, la primera capa tendrá las 15 neuronas (salida del PCA) y la capa final tendrá 11 neuronas, cada una para cada instrumento musical a identificar, en cuanto a la función de optimización se usará Adam con un factor de aprendizaje de 0,00001, se usaron las métricas el error cuadrático medio y accuracy, y para , este modelo tendrá para entrenar 122.423 parametros, a continuación una imagen que resume al modelo:



Se han obtenido los siguientes resultados:

train	valid	test	tiempo respuesta (seg)
90,36%	90,27%	90,09%	0,06

Imagen 18: Resultados entrenar RNA basico

9.2. Modelo RNA experimental

Este modelo tiene como base lo aprendido del diseño del modelo anterior, pero mirando la documentación de tensorflow se ha encontrado que hay varias otros tipos de capas a usar, se han mantenido el optimizador y métricas, pero se hace un diseño donde se integren algunas de esas capas con el intento de mejorar los resultados anteriores, entre ellas la capa Conv1D, UpSampling1D, MaxPooling1D, GlobalAveragePooling1D y Flatten que no se encontraban en el diseño anterior, para un total de 34 capas y 76.255 parametros, mayormente tiene varios bloques como el siguiente:

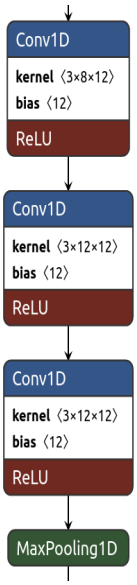


Imagen 19: Bloque dos conv y un maxpooling

Luego del entrenamiento se lograron los siguientes resultados:

train	valid	test	tiempo respuesta (seg)
78,63%	78,72%	80,09%	0,32

Imagen 20: Resultados entrenar RNA experimental

9.3. RandomForestClassifier

Para entrenar el randomForest, básicamente se dejaron las configuraciones por defecto, pues desde el primer entrenamiento se tuvieron resultados bastantes interesantes, por lo que no se tuvo la necesidad de buscar mejorar el modelo, no se muestra el árbol debido a su gran tamaño, mas sólo se muestran los resultados obtenidos:

train	valid	test	tiempo respuesta (seg)
99,81%	95,72%	94,90%	0,79

Imagen 21: Resultados entrenar RandomForest

Si bien los resultados mejoran sustancialmente, el tiempo que toma el modelo en entregar una respuesta a comparación es mucho mayor.

9.4. LogisticRegression

Para entrenar este algoritmo se usa un solver=sag, un máximo de iteraciones de 1000, y activamos la multiclase con ovr, des esta manera el modelo estará listo para entrenarse y al final del entrenamiento nos dará los siguientes resultados:

train	valid	test	tiempo respuesta (seg)
92,18%	92,36%	90,72%	1,86

Imagen 22: Resultados entrenar LogisticRegresosr

9.5. DecisionTreeClassifier

Para el árbol de decisión se ha dejado la probabilidad como None, de esta manera le estamos pidiendo al algoritmo que calcule de forma automática la profundidad, seguidamente le definimos que el mínimo split es de 2 que el

mínimo samples leaf es de 1 y estado randomico es de 123, de esta manera podemos configurar la multiclase en la clasificación.

Luego del entrenamiento se tendran los siguientes resultados:

train	valid	test	tiempo respuesta (seg)
100,00%	92,54%	91,45%	1,15

Imagen 23: Resultados entrenar DecisionTree

Este es el primero que logra al 100% en el train.

9.6. DummyClassifier

Por las mismas características del DummyClassifier se ha optado dejarlo con sus valores por defecto, pues como lo dice la literatura este algoritmo esta diseñado mas con el fin de compara que con el fin de poner en producción, sin embargo, para ser un algoritmo tan poco complejo se obtuvieron resultados bastante aceptable, estos son:

train	valid	test	tiempo respuesta (seg)
85,54%	85,63%	81,54%	3,1

Imagen 24: Resultados del train de DummyClassifier

9.7. KneighborsClassifier

En mi poca experiencia usando este algoritmo he encontrado cierta utilidad en dejar los hiperparametros por defecto, por ello no se considera hacer ningún ajuste de hiperparametros, y se obtienen los siguientes resultados:

train	valid	test	tiempo respuesta (seg)
96,63%	96,72%	96,00%	2,3

Imagen 25: Resultado entrenar KneighborsClassifier

Y como se puede notar, los resultados son no menos que excelentes en términos de métricas de medición.

9.8. GaussianNB

Naive Bayes es un algoritmo particular, cuya forma de aprendizaje es bastante interesante, lo incluí en esta prueba debido a que quería ver que resultados podría dar a comparación con algoritmos mas comunes en aplicaciones de audios.

Los resultados fueron:

train	valid	test	tiempo respuesta (seg)
90,63%	90,81%	89,00%	1,82

Imagen 26: Resultados entrenar Naive Bayes

9.9. Comparación resultados

Para resumir se estipula la siguiente tabla:

Modelos		Algoritmo	train	valid	test	tiempo all test (seg)
Machine learning	Clásicos	RandomForestClassifier	99,81%	95,72%	94,90%	0,79
		LogisticRegression	92,18%	92,36%	90,72%	1,86
		DecisionTreeClassifier	100,00%	92,54%	91,45%	1,15
		DummyClassifier	85,54%	85,63%	81,54%	3,1
		KNeighborsClassifier	96,63%	96,72%	96,00%	2,3
		GaussianNB	90,63%	90,81%	89,00%	1,82
	Deep Learning	Basico 122.000 parametros	90,36%	90,27%	90,09%	0,06
		Experimental	78,63%	78,72%	80,09%	0,32

Tabla 3: Resumen modelos entrenados

9.10. Elección modelo usar

A pesar de que la mayoría de la bibliografía le apuesta a las redes neuronales, para este ejercicio en particular, los algoritmos clásicos, por su sencillez han funcionado bien, claro, esto es luego de aplicar el PCA, antes de ello, ningún de los mejores modelos lograba superar el 82%.

Con esto y teniendo la tabla resumen anterior **se elije el randomForest**, **principalmente por sus porcentajes superiores al 95%** en validación y casi el 95% en el testeo, si bien el KNeighborsClassifier tiene mejores métricas, su tiempo de predicción es muy alto lo que lo hace ineficiente para un entorno de producción. No se elije la red neuronal básica, pues si bien otorga buenas métricas y gran velocidad de respuesta, no son tan altas las métricas como lo es con RandomForest, pero si el tema tiempo de respuesta es un factor clave se podría optar por usar la red neuronal básica).

10. Conclusiones y siguientes pasos

10.1. Conclusiones

- Una buena preparación de los datos hace que incluso un algoritmo sencillo pueda dar buenas métricas de evaluación.
- A pesar de la popularidad de las redes neuronales, no es la mejor opción para todos los problemas, ahí casos donde un algoritmo clásico da mejores resultados por su sencillez.
- Aplicar el PCA ha ayudado mucho para que los modelos pudieran aprender, y esto facilitó el aprendizaje de los diferentes modelos, pero no siempre el PCA tiene estos resultados, así que es podría sugerirse entrenar modelos antes y después del PCA para poder comparar y ver si es útil o no.
- Al inicio del trabajo se esperaba un éxito con un modelo que entregara alguna métrica de evaluación por encima del 85%, pero los resultados obtenidos fueron mejores a lo esperado con métricas por encima del 95%, lo que indica que el método seguido fue correcto.

10.2. Siguientes pasos

- Seguir evaluando el deep learning, esto debido a que una vez se consigan buenas métricas, se puede usar el modelo como transfer learning para problemas futuros de enfoque similar.
- Seguir probando modelos experimentales y usar convoluciones en 2D para comparar los resultados, y quizás hacer una transferencia de conocimiento con algún modelo de los autores de la bibliografía.
- Aplicar el mismo proceso tratado en este trabajo para atacar otros datasets, como por ejemplo Freesound [24]

Bibliografía

1: kaggle, MusicNet Dataset, 2020, https://www.kaggle.com/datasets/imsparsh/musicnet-dataset

2: Music Technology Group - Universitat Pompeu Fabra, music extractor features, 2021

3: Sanket Doshi, Music Feature Extraction in Python, 2018, https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d

4: McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg y Oriol Nieto, librosa: análisis de señales de audio y música en python, 2015

5: Aurora Salgado Díaz del Río, Reconocimiento automático de instrumentos mediante aprendizaje máquina, 2019, https://biblus.us.es/bibing/proyectos/abreproy/92353/descargar_fichero/TFG-2353-SALGADO.pdf

6: John Thickstun, Zaid Harchaoui & Sham M. Kakade, LEARNING FEATURES OF MUSIC FROM SCRATCH, 2017

7: MIT, NSF, , SFE, Lego, scratch, , https://scratch.mit.edu/

8: JUAN SEBASTIÁN MÉNDEZ HERNÁNDEZ, IDENTIFICACIÓN DE INSTRUMENTOS MUSICALES A PARTIR DEL
ALGORITMO MFCC, 2020

9: Juan Sebasti´an G´omez Ca˜n´, Automatic Instrument Recognition using DeepConvolutional Neural Networks, 2018

10: , Masa sonido, Que es la masa del sonido, https://es.wikipedia.org/wiki/Masa_de_sonido

11: , Rmse audios, , https://www.euronics.es/blog/rms/#:~:text=RMS%20(Root%20Mean%20Squared)%20o,calidad%20del%20sonido%20que%20escucharemos)

12: granada, que es espectograma, , https://www.granada.org/inet/sonidos.nsf/d483b298c3f6a1b9c1257cdd00384c53/3fdfc36a7489b607c1257cde0024bb34!OpenDocument#:~:text=El

%20espectrograma%20es%20una%20representaci%C3%B3n,representa%20en%20el%20eje%20horizontal.

13: studio-22, Que es ancho de banda, , https://www.studio-22.com/blog/enciclopedia/ancho-de-banda#:~:text=Ancho%20de%20banda%20%2D%20Referido%20al,es%20mayor%20a%203%20dB)

14: towardsdatascience, extraccion caracteristicas en audio musical, , https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d

15: iberdrola, 2022, https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial

16: IBM, 2019, https://www.ibm.com/co-es/analytics/machine-learning

17: ciencia de datos, 2021, https://www.cienciadedatos.net/documentos/35_principal_component_analysis

18: iartificial, 2018, https://www.iartificial.net/random-forest-bosque-aleatorio/

19: ciencia de datos, 2022, https://www.cienciadedatos.net/documentos/27_regresion_logistica_simple_y_multiple

20: IBM, 2019, https://www.ibm.com/es-es/topics/decision-trees

21: towardsdatascience, 2020, https://towardsdatascience.com/why-using-a-dummy-classifier-is-a-smart-move-4a55080e3549

22: aprendemachinelearning, 2019, https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/

23: programadorclick, 2017, https://programmerclick.com/article/21391478006/

24: Freesound fsd, a dataset of everyday sounds, https://annotator.freesound.org/fsd/

25: Museo Chileno de Arte Precolombino, Clasificación Sachs-Hornbostel de instrumentos musicales: una revisión y aplicación desde la perspectiva americana, 2021, https://www.scielo.cl/scielo.php?pid=S0716-27902013000100003&script=sci_arttext&tlng=pt

26: MANUEL ENRIQUE ALDANA SÁNCHEZ y JUAN SEBASTIAN PERDOMO MÉNDEZ, RECONOCIMIENTO DE PARTITURAS MUSICALES POR MEDIO DE VISIÓN ARTIFICIAL, 2013 https://repositoriousco.co/bitstream/123456789/931/1/TH%20IE%200188.pdf

27: Silvia Jiménez Gómez, GENERACIÓN Y EVALUACIÓN DE SECUENCIAS MELÓDICAS MEDIANTE INTELIGENCIA ARTIFICIAL, 2018, https://oa.upm.es/53396/1/TFG_SILVIA_JIMENEZ_GOMEZ.pdf

.