

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:

Rodríguez Villamizar, William Steve

D.N.I:

1098685961

Director:

Escrig Pérez, Helio

Convocatoria:

Segunda

Orientación:

Virtual

Créditos:

12 ECTS

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 límites.

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 más 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 93% 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 93% 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

Agradecimientos	2
Resumen.....	3
Abstract	4
Tabla de Contenido	5
Índice de tablas	7
Índice de imágenes.....	8
1. Objetivos	10
2. Introducción	12
2.1. Estado del arte.....	12
2.2. Dataset	14
2.3. Metodología CRISP-DM	16
3. Inteligencia artificial	18
3.1. ¿Qué es IA?.....	18
3.2. Que es Machine learning.....	18
3.3. Que es Deep learning	18
3.4. Que es aprendizaje supervisado	19
3.5. Que es PCA	20
3.1. Que es Árbol de decisión.....	21
3.2. Que es Random Forest	22
3.3. Que es Regresión Logística	23
3.4. Que es el clasificador Dummy	24
3.5. Que es Vecinos más cercanos.....	25
3.6. Que es Gaussian Naive Bayes.....	25
4. Preparación de los datos	27
4.1. Metodología a aplicar.....	27
4.2. Metodología Split segundos para facilitar despliegue en producción.....	27
4.3. Data audio incrementation	29
4.3.1. Librería para audio data incrementation	30

4.4.	Extracción características	30
4.4.1.	Taza cruce por cero	31
4.4.2.	Centroide espectral	32
4.4.3.	Reducción espectral	32
4.4.4.	RMS	32
4.4.5.	croma	32
4.4.6.	ancho de banda	32
4.4.7.	MFCC — Coeficientes centrales de frecuencia Mel	32
4.4.8.	Resumen extracción	32
4.4.9.	Librería publicada para extracción de características	33
4.5.	PCA	34
4.6.	Definición configuraciones de datos para modelo.....	36
4.6.1.	Elección del Split en segundos a usar	36
4.6.2.	Función de evaluación a usar	37
4.6.3.	División de datos en train, valid y test	38
5.	Entrenamiento modelo	39
5.1.	Modelo RNA básico	39
5.2.	Modelo RNA experimental	40
5.3.	Entrenando Random Forest	42
5.4.	Entrenando Regresión Logística	42
5.5.	Entrenando el Árbol de decisión	43
5.6.	Entrenando el Clasificador Dummy	43
5.7.	Entrenando el algoritmo Vecinos más cercanos	43
5.8.	Entrenando Gaussiano Naive Bayes	44
5.9.	Comparación resultados.....	44
5.10.	Elección modelo usar	45
6.	Despliegue producción en modo API	46
7.	Conclusiones y siguientes pasos.....	48
7.1.	Conclusiones.....	48

7.2. Siguietes pasos	48
Bibliography.....	49

Índice de tablas

Tabla 1: Resumen dataset	15
Tabla 2: Vector de salida luego de extraer las características del audio	33
Tabla 3: Resumen modelos entrenados.....	45

Índice de imágenes

Ilustración 1. Ejemplo partitura.....	14
Ilustración 2. Ejemplo de red neuronal	19
Ilustración 3. Aprendizaje supervisado	20
Ilustración 4. Ejemplo Árbol de decisión	22
Ilustración 5. Ejemplo Random Forest	23
Ilustración 6. Ejemplo regresión logística.....	24
Ilustración 7. Ejemplo matriz de confusión	24
Ilustración 8. Ejemplo Vecinos más cercanos	25
Ilustración 9. ejemplo de Naive Bayes	26
Ilustración 10. Cantidad de datos según Split elegido	28
Ilustración 11. Cantidad de entradas según Split.....	29
Ilustración 12. tres dimensiones de un audio	31
Ilustración 13. ejemplo cruce por cero	31
Ilustración 14. Librería Python: ProcessAudio.....	33
Ilustración 15. Cantidad de información en cada componente	34
Ilustración 16. Porcentaje de información luego de aplicar PCA	35
Ilustración 17. % Información vs componentes en PCA.....	36
Ilustración 18. Aciertos train modelo default	37
Ilustración 19. RNA Simple	40
Ilustración 20. Resultados entrenar RNA básico	40
Ilustración 21. RNA experimental, parte 1	41
Ilustración 22. RNA experimental, parte 2	41
Ilustración 23. Resultados entrenar RNA experimental	42
Ilustración 24. Resultado entrenar Random Forest	42
Ilustración 25. Resultados entrenar Regresión Logística	42
Ilustración 26. Resultados entrenar Árbol de decisión	43
Ilustración 27. Resultados Clasificador Dummy	43
Ilustración 28. Resultados Vecinos más cercanos.....	44
Ilustración 29. Resultados Naive Bayes	44
Ilustración 30. Contenido archivo despliegue con docker-compose	46
Ilustración 31. Imagen en dockerhub.....	47
Ilustración 32. Usando la API.....	47
Ilustración 33. Respuesta de la API	47

1. Objetivos

La finalidad de este proyecto es entrenar un modelo de inteligencia que pueda definir qué 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 específico se usara la base de datos de Musinet (kaggle & Musinet, 2020).

Además de entrenar un modelo que sea funcional para este trabajo, se ha pensado en la fase final de la metodología CRISP-DM (Detalle sobre el mismo en el capítulo 2.3) la fase de llevar el modelo a producción, como deseado que sea lo más cercano al tiempo real, en esto el tamaño (en segundos del audio) debería ser pequeño para procesarlo rápidamente y dar un resultado que se pueda presentar al usuario final, con ello se decide realizar un Split de los datos para no tratar la canción completa sino el mismo audio pero procesado en fragmentos del mismo (los detalles sobre cuál es el valor en segundos del Split están en el capítulo 4.2).

A su vez sobre los audios, luego del Split, se realiza un incremento de audios al realizar diversas transformaciones sobre los audios, multiplicando de esta manera la cantidad de datos, sobre los cuales se procederá a aplicar una extracción de características, según lo propone (Méndez

Hernández, 2020), en esto solo se elegirán unas características a usar, a fortuna (Doshi, Music Feature Extraction in python, 2018) define las características más relevantes y que en su trabajo dieron importantes resultados, y al igual que El, se usará la librería (librosa, 2022), la cual es una librería para extraer características de audios.

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

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

2. Introducción

2.1. Estado del arte

Durante una exhaustiva investigación de trabajos relacionados se ha revisado entre otros el trabajo de Aurora (Salgado Díaz del Río, 2019), donde en su trabajo usa un dataset que contiene audios de instrumentos específicos (violín, piano, etc.), en su trabajo 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 (E. Fonseca, 2017), donde antes de poder clasificar los instrumentos musicales realizan una extracción de características de forma similar a como lo hace Aurora (Salgado Díaz del Río, 2019) y con otras librerías, dando fuerza a que este es el camino a seguir para los objetivos de este trabajo.

Seguidamente se estudió el trabajo realizado por la universidad de Washington (John Thickstun, 2017), donde recopilaron y clasificaron un amplio dataset (kaggle & Musinet, 2020) y entrenaron varios modelos usando scratch (MIT, 2022) y redes neuronales convolucionales, funciones de activación ReLu, maxpooling, entre otras consideraciones logran resultados más que admirables, pero en particular bastante llamativo se nota la importancia sobre como al aplicar el PCA (Rodrigo, Análisis de Componentes Principales, 2021) a conjunto de datos de tipo audio esto facilita de forma crucial en la facilidad del aprendizaje del modelo.

En particular revisando también el trabajo de JUAN SEBASTIÁN MÉNDEZ HERNÁNDEZ (Méndez Hernández, 2020), 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 más detenimiento en el capítulo 4.4.7.

También y profundizando en la forma como (MANUEL ENRIQUE ALDANA SÁNCHEZ, 2013) ataca el problema de clasificación de partituras, lo cual 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 (GOMEZ, 2018) donde consigue crear un modelo de red neuronal capaz de generar audio, si bien, el trabajo que se presenta no trata de generar audio, las capas de las neuronas y funciones de activación que ella expone son de vital importancia cuando en este trabajo se diseñe el modelo RNA (Capítulo 0).

Finalmente se usará lo aprendido por Juan Sebastián Gómez Cañón (Cañón., 2018), que le en su experiencia luego de crear su clasificador de instrumentos musicales usando redes neuronales, confirma por qué 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 viene bien pues solo se tienen 330 audios (muestras), mientras que en todas las menciones anteriores se contaban con más de 5000 muestras, por lo cual más adelante, además de hacer un audio data incrementation con una librería hecha para este trabajo, basada en (Ma, nlpaug, 2022), se usará una técnica para aumentar la cantidad de muestras mediante un Split de datos, Split que más que ayudar al incremento de datos abre la puerta para usar el modelo en un entorno productivo casi en tiempo real al procesar los audios en secciones de un audio original e ir clasificándolo poco a poco, para por ejemplo una clasificación en una orquesta sinfónica, aunque este ejemplo es un poco ambicioso (por la cantidad de instrumentos superior a los usados para el entrenamiento del modelo) sirve bien para explicar la razón de este Split de datos.

La propuesta del Split de datos si bien no es una idea tomada de ningún estado del arte encontrado, surgió como idea luego de profundizar en el trabajo de (Chris Duxbury, 2001) donde usan técnicas de multirresolución para análisis de audios e información musical en audios cortos de tiempo.

En otros trabajos relevantes se debe mencionar el aporte realizado por (Fu, Lu, Ting, & Zhang, 2010) donde los autores realizan una clasificación de audios con el fin de anotar género, estado de ánimo, reconocimiento de artistas y de instrumentos, si bien su modelo es bastante potente es a la vez un modelo lento para lanzarlo a producción en un hardware de escasos recursos también es importante el tratamiento previo de los datos que en su trabajo demostró grandes rendimientos al modelo que ellos entrenaron.

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 diversos experimentos que fueron plasmados en el repositorio que contiene el código (WISROVI, 2022) y resultados de este trabajo, en el cual se usó la metodología CRISP-DM.

2.2. 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 Kakade (John Thickstun, 2017), donde usaron el dataset de Musinet (kaggle & Musinet, 2020) 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 (kaggle, 2022) la cual 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 musinet (kaggle & Musinet, 2020) .



Ilustración 1. Ejemplo partitura

Fuente kaggle

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 mediante 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

Fuente Musinet (kaggle & Musinet, 2020)

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 se usará como referencia en el momento de realizar el train_test_split con este dato en el momento de preparar los datos (Capítulo 4.6.3), esto debido a que hasta el momento solo se tienen 330 registros de diferentes autores, que si bien es un número de composiciones que en sí, es un valor importante, para entrenar nuestros futuros modelos se va a incrementar la cantidad de registros con audio incrementación y Split de datos la cual es 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.

2.3. Metodología CRISP-DM

Antes de empezar a atacar los objetivos de este trabajo se debe 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 que viene perfecto para este problema.

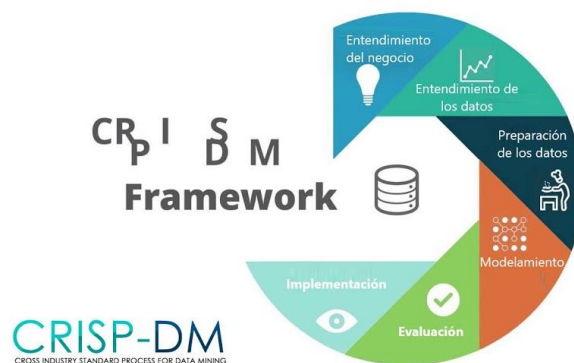


Ilustración 2. Metodología CRISP-DM

Fuente (ytimg, 2022)

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 más 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.

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. ¿Qué es IA?

La Inteligencia Artificial (IA) es la combinación de varios algoritmos planteados con el propósito de crear máquinas que presenten las capacidades iguales o parecidas a las del ser humano, en términos prácticos es enseñarle a la máquina a través de datos ciertas habilidades de análisis que han sido el fuerte del ser humano. En particular esta es una tecnología que todavía nos resulta lejana de una máquina consciente, pero muy real en ejecución de tareas frecuentemente luego de un entrenamiento previo.

3.2. Que es Machine learning

La programación clásica es cuando un usuario programa instrucciones específicas para llevar a cabo una tarea, pero existen infinidad de problemas donde es difícil o casi imposible pensar todos los caminos que debe llevar un proceso para dar la salida correcta, sobre todo si la cantidad de datos es tal que si incrementa la secuencia de instrucciones, se debería modificar el algoritmo para entender ahora los nuevos datos, en esto nace una programación dinámica a los datos de entrada, y se construye un algoritmo con técnicas avanzadas que sin importar los datos de entrada, generalice la información y sobre esta defina un conjunto de caminos a seguir para dar una respuesta, en esto, con un entrenamiento previo, un algoritmo podría recibir un dato y entregar una respuesta consecuente al mismo, en esto se dice que el algoritmo aprende de los datos en lugar de aprender de la programación.

Para definir el machine learning prefiero usar la definición de IBM ya es muy completa “Machine learning es una forma de la IA que permite a un sistema aprender de los 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

Si bien el machine learning es un conjunto de algoritmos que tratan de replicar procesos que repliquen algunos eventos de la naturaleza (generalmente), cuando en esa naturaleza llegamos al ser

humano para replicar su procesamiento cerebral, nace la necesidad de un algoritmo que pueda procesar la información como lo haría el ser humano, basado en su red neuronal, y de esta manera nace una red neuronal artificial, en el cual un conjunto de varias neuronas sería lo que conoceremos como una capa de red neuronal, y el conjunto de capas sería una red neuronal, el problema de una red neuronal artificial al igual que una natural es su complejidad que para verla se requiere un microscopio, pero como no todo el mundo tiene un microscopio, entonces se centra la atención en la capa primera y la capa final, y en ello a las capas intermedias se conocerán como capas profundas, de ahí que se conozca como un aprendizaje profundo o Deep learning (por sus palabras en inglés).

El Deep learning es especialmente útil cuando se trata de aprender patrones de datos no estructurados. Estas están diseñadas para emular de forma muy similar el cómo funciona el cerebro humano. El Deep learning es usado popularmente para reconocimiento de imágenes, voz y aplicaciones de visión de computadora, sin embargo, puede ser usado para atacar cualquier problema que requiera una modelación de datos, esto es más práctico verlo a través de una imagen:

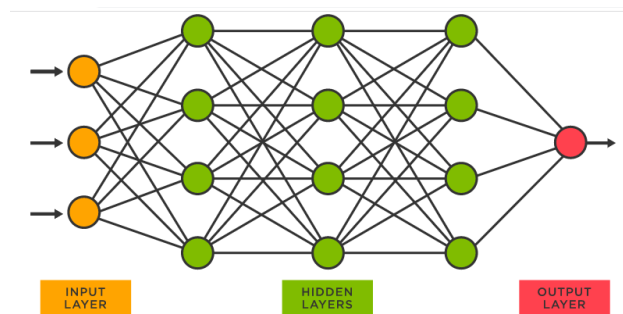


Ilustración 3. Ejemplo de red neuronal

Fuente (tibco, 2022)

En la imagen anterior se puede observar cómo están interconectadas las neuronas, en especial lo que se conoce como capa de entrada (que contiene las neuronas que reciben los datos de entrada), la capa de salida (con la o las neuronas que dan la respuesta luego del procesamiento de los datos) y las capas ocultas (estas pueden ser una o varias, y son las encargadas de dar el procesamiento de los datos mediante transformaciones de los mismos), las líneas que unen las neuronas es como cambian información 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 cuestión.

3.4. Que es aprendizaje supervisado

En términos de IA hay dos formas de generalizar el conocimiento sobre los datos, cuando se tiene las salidas esperadas (por ejemplo, esta imagen es un gato, esta imagen es un perro, etc.) y en base a estas salidas se desea entrenar un modelo, se dice que esta supervisado este entrenamiento, y

cuando no se tienen dichas etiquetas el aprendizaje será no supervisado, pero este último no se tocará en este documento.

Este tipo de aprendizaje tiene la intención de encontrar patrones en los datos para realizar una analítica sobre los mismos, estas etiquetas son lo que define el significado del dato de entrada, por ejemplo si se entrena un modelo que distinga entre imágenes de gatos e imágenes de perros, se tendrán un conjunto de imágenes de gatos y perros, pero es imposible tener todas las imágenes posibles de imágenes de gatos y perros (edad, tamaño, raza, ángulo de toma de la foto, calidad de imagen, cercanía del animal a la cámara que toma la foto, posición del animal, etc.), en ello en lugar de poder tener todas las posibles imágenes, se limita a un pequeño conjunto de dichas posibles imágenes, y se entrena un modelo de forma tal que aprenda la generalidad de los datos y cuando se le entregue una imagen nueva que es diferente a todas las imágenes usadas previamente para entrenar el modelo, el conocimiento del modelo le permitirá predecir con una probabilidad a que categoría es la nueva imagen (por ejemplo, es 85% un gato y 15% un perro, en esto suponiendo que el umbral de decisión es superior al 80%, entonces el modelo habrá predicho que es un gato la nueva imagen presentada al modelo para predecir), en esto en base a esta probabilidad definir una respuesta.

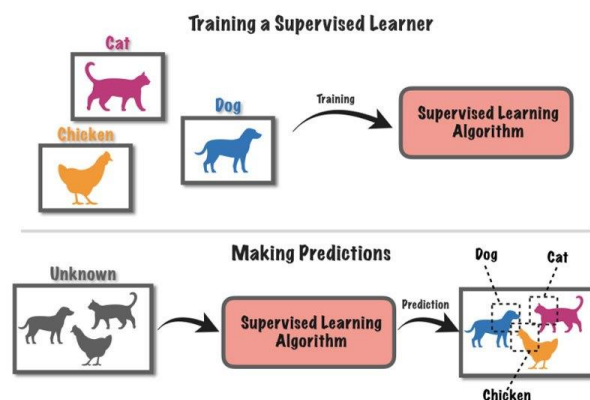


Ilustración 4. Aprendizaje supervisado

Fuente (Apren. Super, 2022)

3.5. Que es PCA

Un dataset tiene una cierta cantidad de entradas que serán las entradas del modelo, para un modelo computacional, entre menos entradas hayan más fácil el modelo podrá aprender la generalidad de los datos y en consecuente dar una predicción más acertada posible, pero al tener una cantidad de entradas y definir cuales entradas son más importantes que otras es un proceso complejo, para ello una técnica de aprendizaje no supervisado llamado PCA (análisis de componentes principales) evalúa

la correlación entre las entradas y proporciona el porcentaje de información presente en cada entrada, en ello, ya se podrá definir cuales entradas son porcentualmente más representativas que otras y cuales entradas no lo son, esto no se realiza por el hecho de eliminar aquellas entradas que menos referentes sean, sino para ilustrar que al usar algunas entradas de las entradas originales, se está manteniendo un porcentaje de la información original, y con ello definir (de forma manual) que porcentaje de información se desea mantener para reducir la cantidad de entradas original a una cantidad de entradas menor que favorezca al modelo su aprendizaje.

Seguramente esta es la cuestión importante pues sacrificar un porcentaje de información para reducir la cantidad de entradas, aunque favorezca al modelo, genera la pregunta: ¿Qué porcentaje es adecuado sacrificar?, realmente esta pregunta no tiene una respuesta que se pueda generalizar para todos los problemas, sencillamente se puede decir que a menos que se evalúen todas las opciones y los datos sobre este porcentaje en PCA permitió que el modelo tenga una validación del modelo en un x% y este otro porcentaje de PCA haga que el modelo se valide con un y%, entonces se podrá definir cuál porcentaje de PCA es mejor que otro porcentaje, y elegir en consecuencia ese porcentaje de PCA que es mejor de acuerdo a los resultados obtenidos, ahora hacer este recorrido requiere mayor tiempo, mayor costo computacional y más diseño del código que realice este recorrido y almacene los resultados, por ello no se suele hacer este recorrido por todos los porcentajes por haber sino elegir algunos (aquellos prometedores) cuya relación entre porcentaje de información reducida y cantidad de nuevas entradas tenga una relación que se pueda considerar como “un buen negocio”, por ejemplo, si ante unos datos, sacrificar el 7% de la información permite reducir la cantidad de entradas a la mitad (reducción al 50%), es decir se tendría una relación 7/50, que al resolver sería 0.14, entre menor del resultado, será mejor el negocio, y en esto se elige este valor de PCA usando el porcentaje de información mantenida, en el ejemplo usado sería un PCA con configuración del 93%.

Ya habiendo elegido este valor, se le configura al PCA y se le entregan los datos de entrenamiento, de esta manera el PCA podrá simplificar la cantidad de entradas a la vez que conserva la mayor información posible, de esta forma se condensa la información en las nuevas entradas, entradas que son diferentes a las entradas originales, por tanto no se ha realizado una eliminación de algunas entradas sino se ha tomado todas las entradas y se han transformado para generar las nuevas entradas resultantes, y esto lo hace un método bastante útil para aplicar en algoritmos de inteligencia artificial.

3.1. Que es Árbol de decisión

Un árbol de decisión es un algoritmo de aprendizaje supervisado cuyo pseudocódigo es similar a las hojas de un árbol y sus caminos desde el tallo principal, donde la raíz recibe los nutrientes y estos siguen un camino para llevar a las hojas cada nutriente de forma individual, en términos prácticos estos nutrientes son los datos, pero se procesan de uno en uno, a medida que un dato va moviéndose por las ramas del árbol se van tomando decisiones en los nodos donde se presentan dos o más

caminos con el objetivo de elegir un solo camino (rama), sin retroceder y siempre hacia adelante hasta llegar a la hoja decidida, y estas decisiones definen un único camino tomado desde la raíz hasta la hoja, donde el camino que sigue es único, ahora estas hojas representan las salidas, por ejemplo si el dato fuera una imagen de una vaca y se tuvieran que clasificar entre 10 animales de una granja, se tendrían 10 hojas en este árbol de decisión, el tallo recibiría la imagen de la vaca y a través de varias preguntas este dato viajaría por diferentes ramas del árbol hasta llegar a la hoja correspondiente y definir que es una imagen de una vaca. Un ejemplo sería:

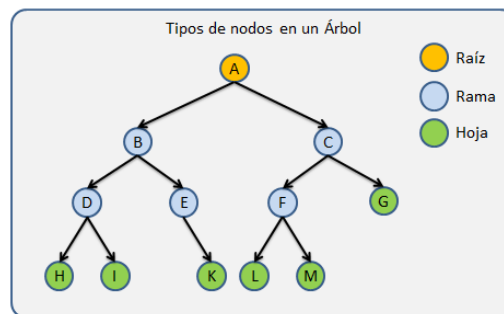


Ilustración 5. Ejemplo Árbol de decisión

Fuente (ArbolDecision, 2022)

Si bien el ejemplo usado es de clasificación, que es cuando se etiqueta el dato en alguna categoría (por ejemplo: categoría de vaca), los árboles de decisión sirven también para la regresión que es cuando en lugar de definir una categoría para un dato, el árbol puede definir una proyección de un dato, por ejemplo: el futuro valor del dólar en base a movimientos del valor del dólar en el tiempo en el último mes.

3.2. Que es Random Forest

El Random Forest (bosque aleatorio en español) es en sí, un conjunto de árboles de decisión que forman equipo cuya interrelación es por bagging (bagging, 2022), y por ende también hace parte de los algoritmos de aprendizaje supervisado, por ejemplo un bosque podría tener 4 árboles, cada árbol funcionaría tal cual como lo explicado en el Capítulo 3.1, como se tienen varios árboles, no es necesario que todos los árboles reciban todos los datos sino que cada uno recibe una porción de los mismos y cada árbol daría su predicción a distintas muestras del mismo problema, con esto al combinar las respuestas de los árboles algunos errores se compensan con otros y la predicción que generaliza mejor la respuesta a los datos, además de que cada árbol tiene menos datos para entrenar entonces su entrenamiento es más rápido, a la vez que se puede entrenar cada árbol en paralelo, una imagen representativa sería la siguiente:

Random Forest Classifier

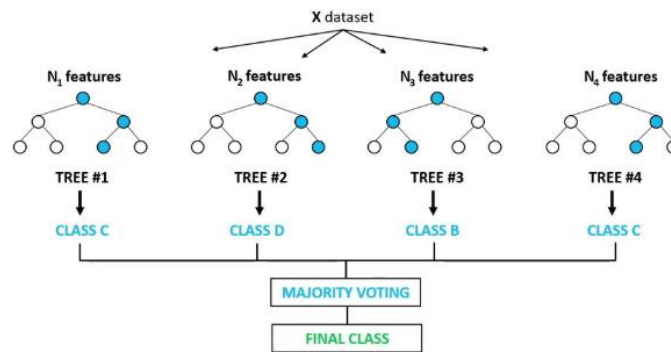


Ilustración 6. Ejemplo Random Forest

Fuente (Ejemplo RandomForest, 2022)

3.3. Que es Regresión Logística

La regresión es un tipo de algoritmo donde se estima el valor de probabilidad de que una respuesta sea dada a una entrada, en términos de clasificación esto generara un valor de probabilidad para cada categoría donde sumando todos los valores de probabilidad da 1, en esto para usarla ante un problema de clasificación lo que se realiza es una aproximación binaria al dígito más cercano (cero o uno) a partir de un umbral mínimo de probabilidad, es decir si para una categoría X se tiene una probabilidad del 0.85 y el umbral de decisión de aproximación binaria es de 0.6, entonces se aproximara diciendo que esa entrada pertenece a la categoría que se aproximó y las demás categorías serán aproximadas a cero indicando con esto que pertenece a una categoría y a las demás no, una imagen que lo podría representar seria:

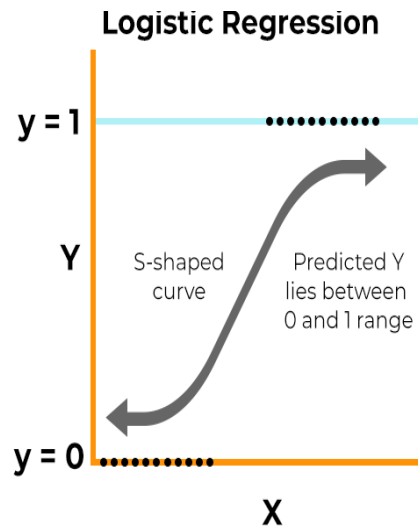


Ilustración 7, Ejemplo regresión logística

Fuente pimages (pimages, s.f.)

3.4. Que es el clasificador Dummy

El clasificador Dummy es quizás uno de los más sencillos y es debido a contrario a otros algoritmos de aprendizaje supervisado, este no intenta aprender la generalidad de los datos o patrón de los mismos, este modelo analiza cual etiqueta es más frecuente en el conjunto de datos de entrenamiento y hace algunas predicciones basadas en esta etiqueta, por ello a veces se le llama clasificador ficticio y se le suele representar con una matriz de confusión, así:

		Ground Truth	
		Positive	Negative
Prediction	Positive	True positives	False positives
	Negative	False negatives	True negatives

Ilustración 8. Ejemplo matriz de confusión

Fuente (interactivechaos, 2022)

3.5. Que es Vecinos más cercanos

El clasificador por agrupamiento de vecinos más cercanos es un algoritmo de aprendizaje supervisado donde se busca clasificar los datos en base a similitud con otros datos del dataset, de esta manera un nuevo dato será clasificado acorde al grupo de datos (categoría) más cercana, de esta manera se establecen cantidad de grupos iguales a la cantidad de categorías, y el algoritmo aprende a agrupar los datos durante el entrenamiento, de esta manera cuando llega un nuevo dato, este es agrupado a la categoría más próxima en el momento de realizar la predicción, por ejemplo:

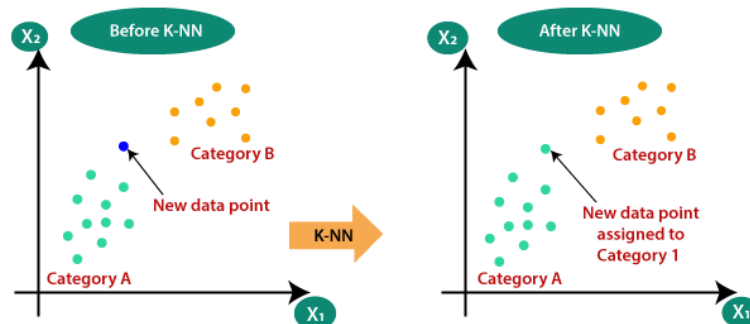


Ilustración 9. Ejemplo Vecinos más cercanos

Fuente (javatpoint, 2022)

3.6. Que es Gaussian Naive Bayes

El clasificador Gaussiano de Naive Bayes, es un algoritmo de aprendizaje supervisado, pero es diferente a los demás, a diferencia de los algoritmos anteriormente mencionados los cuales son discriminativos a las categorías, donde un dato pertenece a solo una categoría, es decir, aprenden directamente la relación entre la salida de los datos y los datos en sí. En lugar de hacer esto Naive Bayes se basa en la probabilidad condicional y el teorema de Bayes.

Esta probabilidad condicional es establecer la probabilidad en base al resultado actual (llamaremos a esta probabilidad como A para fácil interpretación) y a los resultados previos (llamaremos a esto B para poder entenderlo más fácil), es decir tomando la probabilidad del evento previo y la probabilidad del evento actual se calcula una nueva probabilidad en base a estos dos datos así: $P(A | B) = \frac{P(A, B)}{P(B)}$, pero esta ecuación es fácil cuando se habla de solo dos eventos, cuando son muchos eventos, todos independientes entre sí en términos de que cada entrada no tiene relación entre ellas mismas, o al menos se asume que no tienen relación y es por esta suposición que a veces a este algoritmo también se le conoce como Bayes simples, se utiliza el teorema de Bayes:

$$P(B_1, B_2, \dots, B_n, A) = P(B_1 \mid B_2, B_3, \dots, B_n, A)P(B_2, B_3, \dots, B_n, A)$$

Por supuesto no se trata de resolverla de forma manual y menos cuando en un conjunto de entrenamiento puede haber fácilmente unos 10.000 datos para ingresar a esta ecuación, si bien no es compleja, resulta ser, este algoritmo, un algoritmo muy potente. Un ejemplo sería:

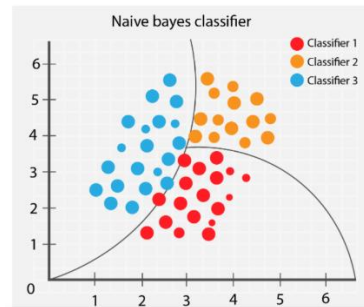


Ilustración 10. ejemplo de Naive Bayes

Fuente (analyticsvidhya, 2022)

4. Preparación de los datos

4.1. Metodología a aplicar

En este apartado se define el proceso a seguir para preparar los datos, para ello y considerando la experiencia de Aurora Salgado (Salgado Díaz del Río, 2019), 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 revisando otras fuentes bibliográficas con el fin de comparar y hallar el factor común entre las diversas metodologías, para ver cuáles son las características más relevantes, con el trabajo de Sanket Doshi (Doshi, Music Feature Extraction in Python, 2018) 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 Sebastián Méndez (Méndez Hernández, 2020), fortalece algunos de ellos y enfoca un valor que resalta la importancia del MFCC.

Si bien Juan Sebastián usa mayormente el MFCC, Sanket Doshi y Aurora tiene un punto en común con las características mencionadas, con ello se tienen 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 (José Pérez de Arce, 2013) a esas 26 posiciones del vector resultante le aplicaremos el PCA para reducir aún más la dimensionalidad de este vector.

4.2. Metodología Split segundos para facilitar despliegue en producción

Hasta el momento se tienen los datos en crudo proporcionados por kaggle, pero la mayoría de los audios tienen una duración superior a los 2 minutos y casi todos en tiempos distintos, por ello se requiere estandarizar la información de entrenamiento buscando que todos tengan la misma longitud temporal por ello se decide no usar el audio con su total longitud sino procesar el audio por fracciones del mismo en modo similar a lo que es un espectrograma, con esto se estará directamente aplicando el otro punto a considerar el cual viene relacionado con lo expuesto por Juan Sebastián Gómez Cañón (Cañón., 2018), 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 al realizar este Split de los datos se estaría haciendo un ligero incremento en la cantidad de datos a comparación del tamaño original.

Ahora para tomar la elección sobre cuál es el Split correcto, se ha calculado el cómo afecta el incremento de datos a medida que se hace el Split, como la idea de este modelo es poder aplicar todas las etapas de la metodología CRISP-DM (Capítulo 2.3), incluyendo la etapa de llevar el modelo a producción, puntualmente en un entorno donde se pueda implementar el modelo casi en real time, esto se puede representar así:

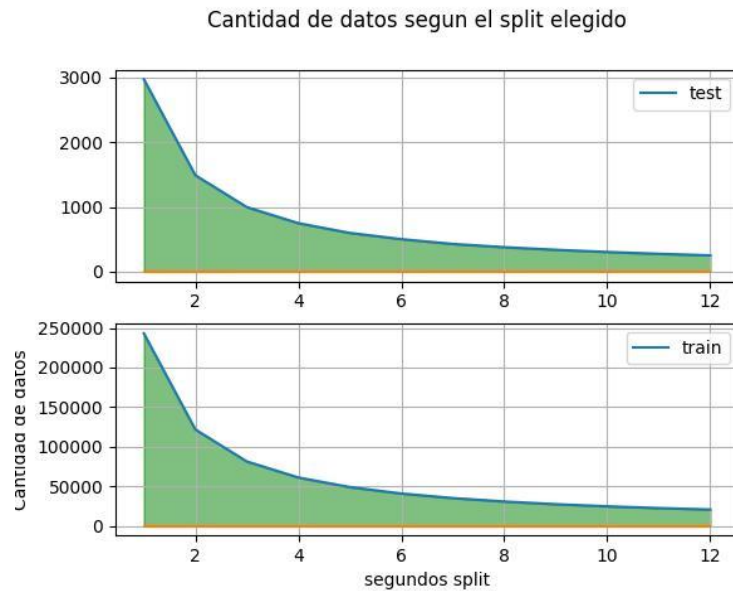


Ilustración 11. Cantidad de datos según Split elegido

Fuente: autor

Por supuesto al fragmentar el audio también se debe fragmentar el vector de salida (con las etiquetas que apliquen) esto para cada fragmento de audio, por ejemplo, si se divide en secciones de 2 segundos un audio de 100 segundos se tendrán 50 muestras de audio y sus 50 salidas.

En total hay 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.

Esta división también afecta el tamaño del vector que representa cada audio (antes de la extracción de características que se aplicará más adelante), si bien esto no afecta el tamaño del vector que se entrega al modelo, entre más corto sea el vector más rápido será la extracción de características, una relación se vería con el siguiente gráfico:

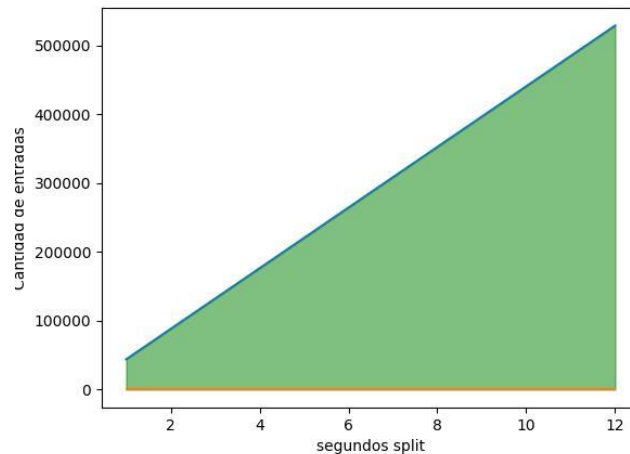


Ilustración 12. Cantidad de entradas según Split

Fuente: autor

Con lo cual ya se tienen dos relaciones: la primera es inversa donde a un Split pequeño (ejemplo 2 segundos) se tienen un aumento de datos significativo y dos, directa, es que a un Split pequeño el vector del audio será pequeño y con ello pequeño el vector que lo representa.

En este punto de este trabajo aún no se definirá el valor del Split a usar, esto se hace en el apartado de modelado (más puntualmente en el capítulo 4.6.1), por lo que por ahora centraremos la atención en realizar data incrementation sobre cada fragmento de audio.

4.3. Data audio incrementation

En el momento de necesitar realizar un incremento de datos ahí varias transformaciones que se le pueden realizar a los datos, desde poner ruido, hacer modificaciones a la frecuencia o velocidad de reproducción, entre otras, afortunadamente existe la librería `nlpaug` (Ma, `nlpaug`, 2019), la cual es una librería para realizar diversas transformaciones sobre audios entre ellas están:

- Añadido de ruido aleatorio
- Añadido de una máscara para poner elementos de fondo
- Pitch para ajuste de tono
- Añadido ruido de fondo en base a armónicos del mismo audio
- Modificación parcial de volumen sobre un tramo del audio

- Cambio de tono parcial en una porción del audio
- Ajuste volumen del audio
- Cambio velocidad de reproducción del audio
- Normalización sobre la máscara de audio

Todas las transformaciones sobre los audios no se pueden realizar en todas las ocasiones, por ejemplo, si en un audio abunda silencio (no sonidos) en una porción del audio, esto genera un error para algunas conversiones, pero las demás se podrán realizar con normalidad, por ello se han aplicado la mayoría de las conversiones según el audio lo permita.

En esto, casi se tendrá un aumento de data en una relación de 1 a 9, es decir un audio se puede convertir a un máximo de 9 audios, los cuales ya estarán listos para el siguiente paso, extracción de características.

4.3.1. Librería para audio data incrementation

***NOTA: falta publicar la librería
de Python y poner el link acá***

4.4. Extracción características

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, específicamente en este trabajo el de clasificación.

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.

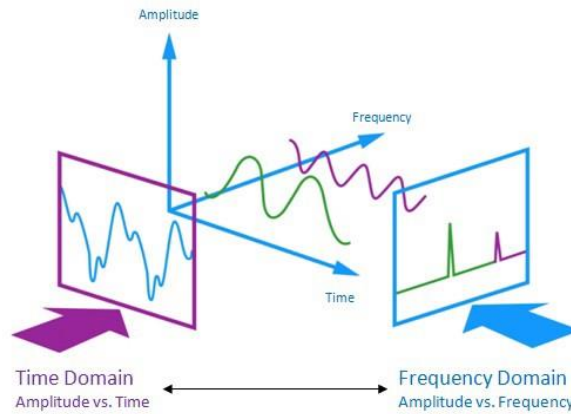


Ilustración 13. tres dimensiones de un audio

Fuente (towardsdatascience)

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

Se usará Librosa (librosa, 2022), una librería de Python para analizar y extraer características de una señal de audio

4.4.1. 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 cinco cruces por cero:

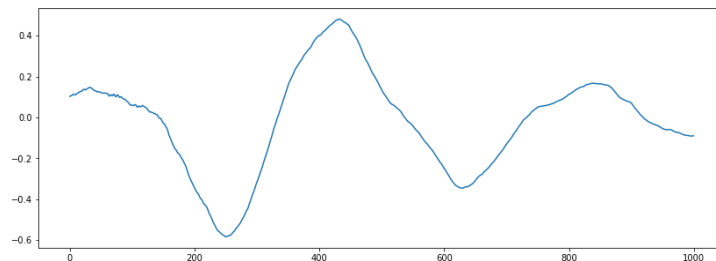


Ilustración 14. ejemplo cruce por cero

Fuente (joserzapata, 2022)

4.4.2. Centroide espectral

Este inicia donde se encuentra el “centro de masa” 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 Centroides espectral estaría alrededor de un centro.

4.4.3. Reducción espectral

Es la frecuencia por debajo de la cual se encuentra un porcentaje específico de energía espectral, dicha energía es el promedio de la energía del audio.

4.4.4. 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.

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

4.4.6. 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 señal.

4.4.7. 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 espectral. Prácticamente el audio original se divide en fracciones (generalmente 20), de esta manera se en cada sección se halla la medida de la envolvente.

4.4.8. 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

Fuente: autor

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.

4.4.9. Librería publicada para extracción de características

Para facilitar esta tarea de extracción de características se ha construido una librería que usando algunas dependencias de procesamiento de audio como Librosa y scipy y junto a lógica propia recibe un audio y le extrae estas 26 características, entregando la respuesta en forma de vector, esta se ha publicado en el nombre de ProcessAudio (wisrovi, ProcessAudio, 2022).

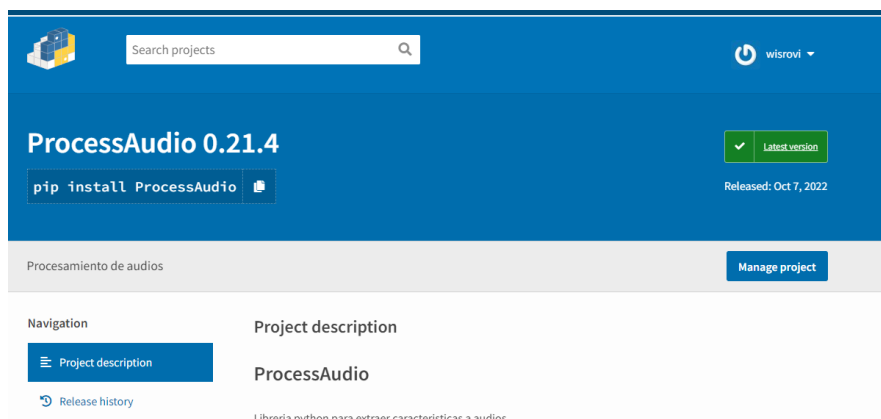


Ilustración 15. Librería Python: ProcessAudio

Fuente: autor

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 el Split de datos, tema que se solucionará en el modelado (Capítulo 4.6.1).

4.5. 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:

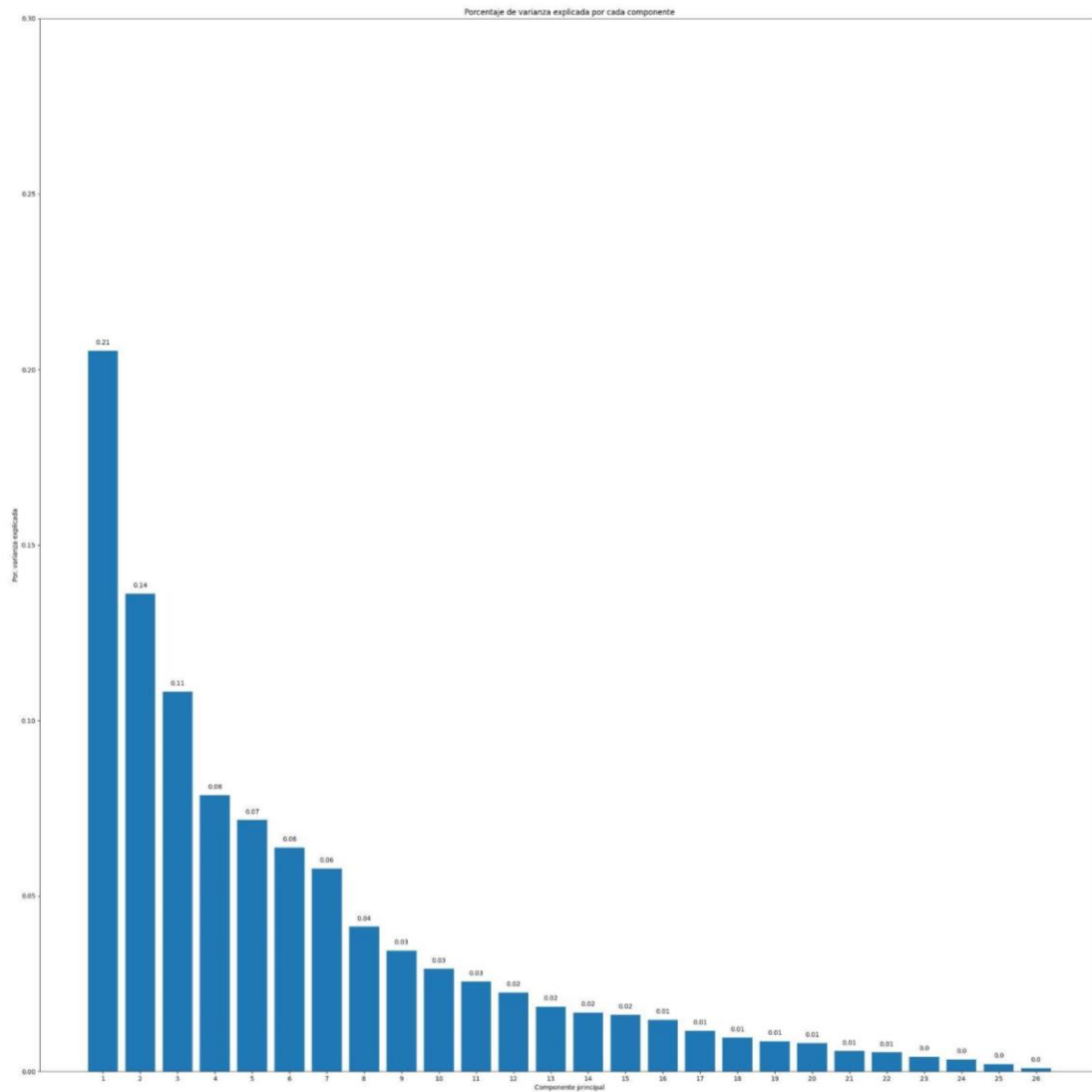


Ilustración 16. Cantidad de información en cada componente

Fuente: autor

Como se puede observar, las ultimas componentes contienen muy poco de la información original, así que con el PCA, lo cual indica que al aplicar el PCA si se logra tener una reducción de componentes,

para ello se evaluará que valor sería más apropiado como nueva cantidad de componentes, en pocas palabras el vector 26 posiciones lo vamos a reducir a un valor menor, por ejemplo 21 componentes, de esta manera al modelo que se entrene le resultará más 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 más de la información, pero cuál sería el porcentaje de la información que se mantendría luego de una reducción de dimensionalidad, para facilitar esta decisión se ha construido el siguiente gráfico:

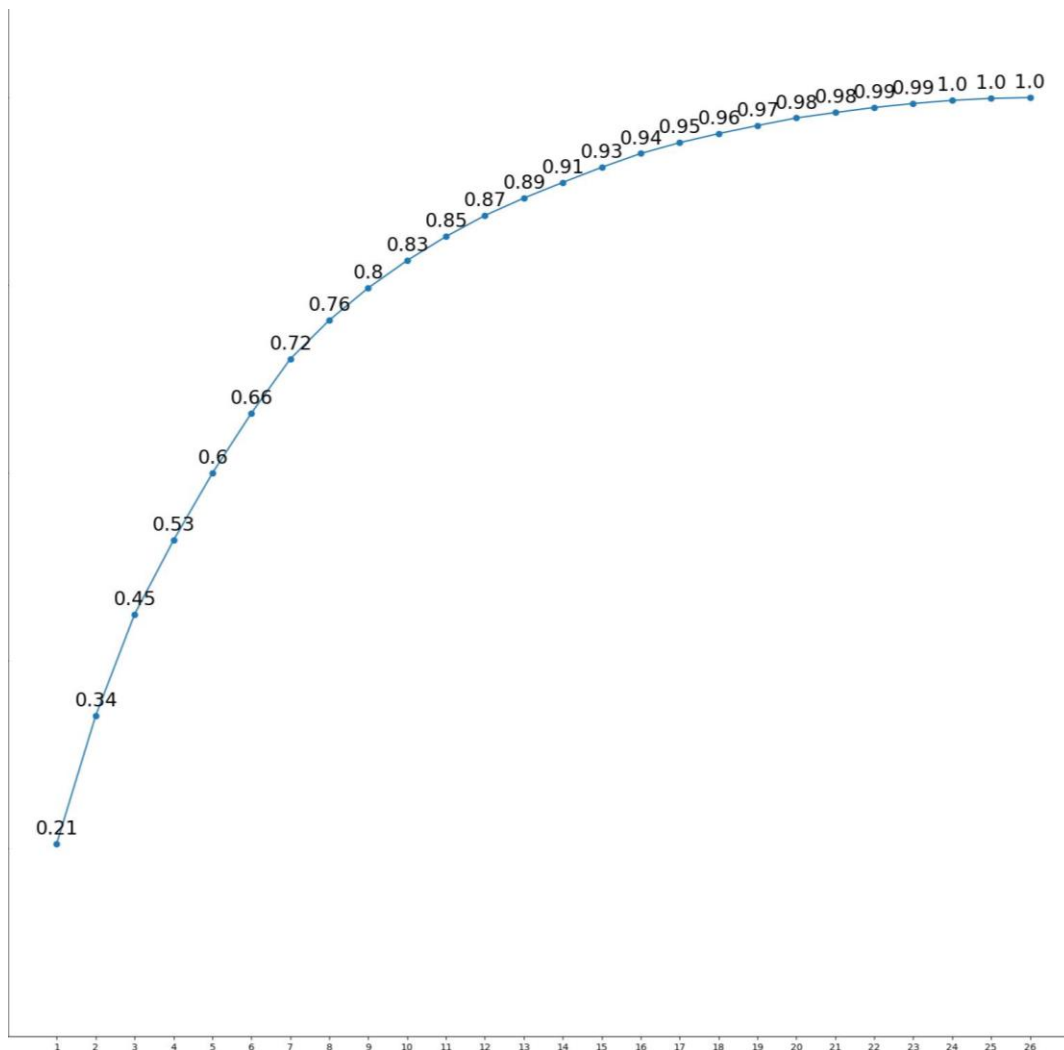


Ilustración 17. Porcentaje de información luego de aplicar PCA

Fuente: autor

Ahora la decisión no es fácil, básicamente se busca sacrificar la menor cantidad de información, pero ganando una buena reducción. Con el 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.

Con este análisis, parece conveniente mantener el 93% de la información puesto que así se logra reducir casi a la mitad la cantidad de entradas, por ello se parametriza para este trabajo el 93% para el PCA.

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				

Ilustración 18. % Información vs componentes en PCA

Fuente: autor

4.6. Definición configuraciones de datos para modelo

4.6.1. Elección del Split en segundos a usar

Ahora ha llegado el momento de tomar la decisión crucial que va a definir cuál es el Split a usar, para ello no se usó una decisión aleatoria o una de alguna bibliografía debido a que esta división surgió durante conversación con el tutor para facilitar el tratamiento de los datos, por lo que se usó el método de evaluación, anteriormente en la ilustración 14, se mostró la posibilidad de usar de 1 a 12 como valor de segundos para el Split, y se profundizó en ese valor en el Capítulo 4.2.

Para poder realizar esta evaluación se entrenaron para cada uno de los Split (1 a 12) un Random Forest con valores por default, buscando ver cómo afecta el aprendizaje del modelo el Split a usar., para ello se muestra el siguiente gráfico que resume un entrenamiento para cada Split:

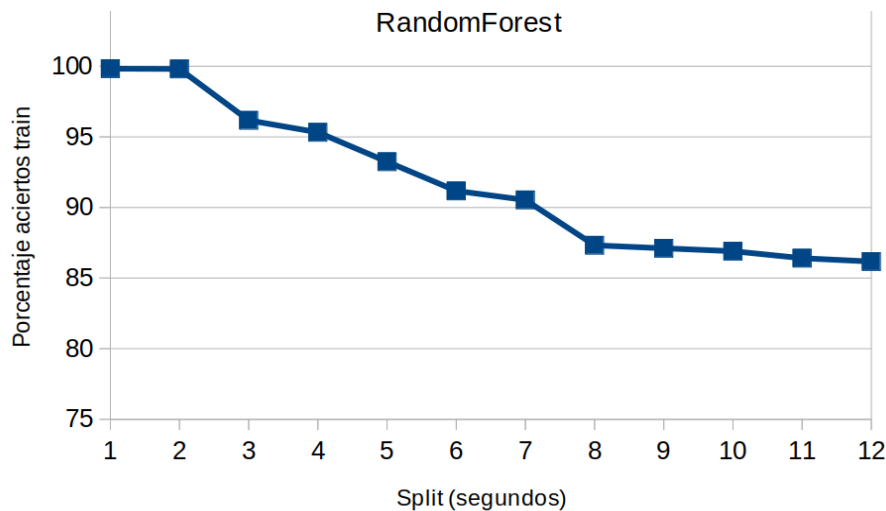


Ilustración 19. Aciertos train modelo default

Fuente: autor

Para el entrenamiento expuesto a cada audio se ha hecho cada Split expuesto y cada vector resultante se aplicó la extracción de características, incremento de datos 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 realizó un train usando el algoritmo Random Forest (con hiperparametros por default) con cada archivo de cada Split y con los resultados de cada entrenamiento se generó el grafico anterior, esto para determinar cuál Split era el más 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%, con ello se nota un reflejo en esos puntos de inflexión que acelera de forma importante el aprendizaje del modelo, por lo que se elige usar el Split de 2 dado que entrega en % de aciertos más alto entre los dos.

4.6.2. 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, aún la matriz de confusión (para multiclase), por ello se creó una función que contabiliza la cantidad de aciertos (verdaderos positivos y verdaderos negativos) y los no aciertos como falsos (falsos positivos y falsos negativos), 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.

4.6.3. División de datos en train, valid y test

Cada dato será 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 $\pm 121700 \times 9$ muestras (el multiplicado a 9 es gracias al incremento expuesto en el capítulo 4.3), 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 se sugiere en estas competencias dejar el set de datos de test para evaluar el entorno productivo y el set de datos de train usarlo para train y valid, generalmente 80% y 20% respectivamente, siendo así se tiene (con Split a 2 segundos):

- Train: 694688 muestras
- Valid: 173672 muestras
- Test: 744 muestras (dadas por kaggle)

5. 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*9 se ha diseñado dos modelos de red neuronal, usando la librería de keras con backend de tensorflow 2.1, que par fin de diferenciarlos se han 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 porque habían tomado esa decisión, por ello se quiso por iniciativa en este trabajo comparar con algunos modelos de inteligencia artificial clásicos y en base a esta comparación determinar el porqué de la decisión de estos autores, en ello se decidió entrenar algunos modelos vistos durante el Máster, estos son: Random Forest, Regresión logística, Árbol de decisión, Clasificador Dummy, Vecinos más cercanos y Gaussian Naive Bayes, y en base a esto comparar los resultados, para estos modelos se usó el MultiOutputClassifier (para las salidas multietiqueta), GridSearchCV (Para varias validaciones de parámetros) y 10-Kfold para entrenar varios modelos del mismo tipo de forma simultanea (con ligeros cambios en los hiperparametros) y quedarnos con el mejor de todos los modelos.

Al final de este capítulo se presenta una tabla resumen comparando los resultados y se elegirá el modelo con las mejores métricas en la comparación, el cual se hará un despliegue como API (Detalles en el capítulo 6).

5.1. Modelo RNA básico

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 parámetros, a continuación una imagen que resume al modelo:

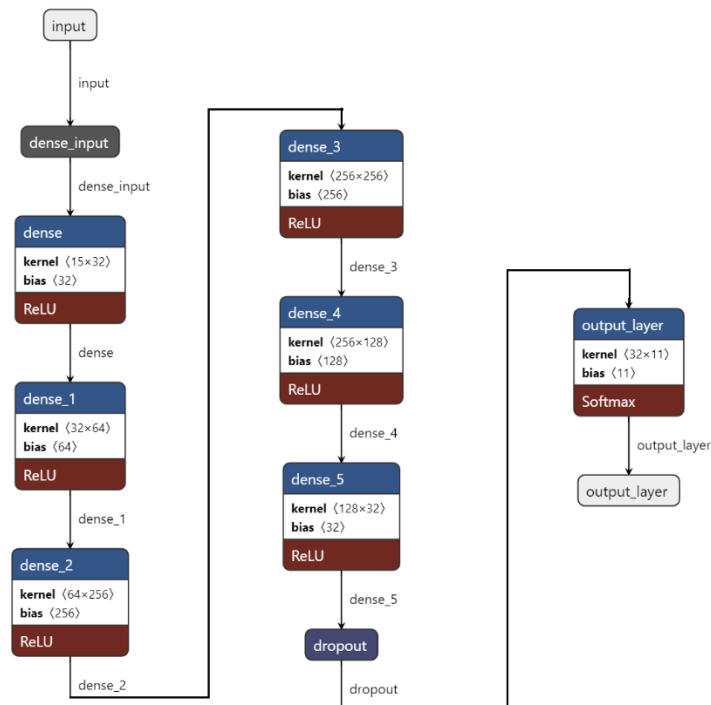


Ilustración 20. RNA Simple

Fuente: autor usando (netron, 2022)

Se han obtenido los siguientes resultados:

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

Ilustración 21. Resultados entrenar RNA básico

Fuente: autor

5.2. 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 parámetros, mayormente tiene varios bloques como el siguiente:

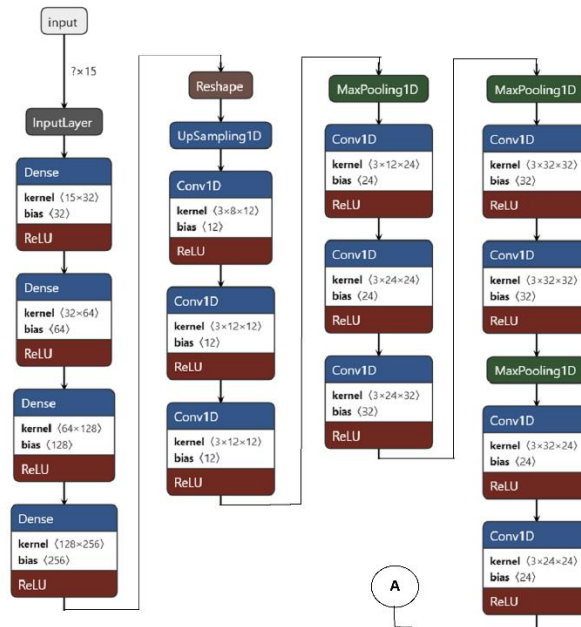


Ilustración 22. RNA experimental, parte 1

Fuente: autor usando (netron, 2022)

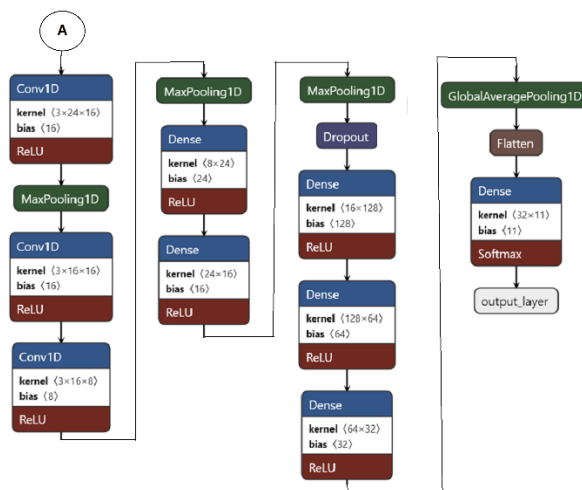


Ilustración 23. RNA experimental, parte 2

Fuente: autor usando (netron, 2022)

Luego del entrenamiento se lograron los siguientes resultados:

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

Ilustración 24. Resultados entrenar RNA experimental

Fuente: autor

5.3. Entrenando Random Forest

Para entrenar el Random Forest, 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

Ilustración 25. Resultado entrenar Random Forest

Fuente: autor

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

5.4. Entrenando Regresión Logística

Para entrenar este algoritmo se usa un solver=sag, un máximo de iteraciones de 1000, y activamos la multiclase con ovr, de 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

Ilustración 26. Resultados entrenar Regresión Logística

Fuente: autor

5.5. Entrenando el Árbol de decisión

Para el árbol de decisión se ha dejado la profundidad 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 Random es de 123, de esta manera podemos configurar la multiclase en la clasificación.

Luego del entrenamiento se tendrán los siguientes resultados:

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

Ilustración 27. Resultados entrenar Árbol de decisión

Fuente: autor

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

5.6. Entrenando el Clasificador Dummy

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

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

Ilustración 28. Resultados Clasificador Dummy

Fuente: autor

5.7. Entrenando el algoritmo Vecinos más cercanos

En mi poca experiencia usando este algoritmo he encontrado cierta utilidad en dejar los hiperparametros por defecto, muy poco he tenido que hacer ajuste de hiperparametros, sin embargo, la validación cruzada se encarga de buscar la mejor combinación de hiperparametros, por ello no se considera hacer ningún ajuste de hiperparametros, pues al final el GridSearchCV finalizo dejando los hiperparametros por defecto, y se obtienen los siguientes resultados:

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

Ilustración 29. Resultados Vecinos más cercanos

Fuente: autor

Y como se puede notar, los resultados son no menos que excelentes en términos de métricas de medición, lo único es el tiempo que toma en realizar una predicción.

5.8. Entrenando Gaussiano Naive Bayes

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

Los resultados fueron:

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

Ilustración 30. Resultados Naive Bayes

Fuente autor

5.9. Comparación resultados

Para resumir, se estipula la siguiente tabla:

Modelos		Algoritmo	train	valid	test	tiempo all test (seg)
Machine learning	Clásicos	Random Forest	99,81%	95,72%	94,90%	0,79
		Regresión logística	92,18%	92,36%	90,72%	1,86
		Árbol de decisión	100,00%	92,54%	91,45%	1,15
		Clasificador Dummy	85,54%	85,63%	81,54%	3,1
		Vecinos más cercanos	96,63%	96,72%	96,00%	2,3
		Gaussiano Naive Bayes	90,63%	90,81%	89,00%	1,82
	Deep Learning	Básico 122.000 parámetros	90,36%	90,27%	90,09%	0,06
		Experimental	78,63%	78,72%	80,09%	0,32

Tabla 3: Resumen modelos entrenados

Fuente autor

5.10. Elección modelo usar

A pesar de que la mayoría de la bibliografía les 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 elige el Random Forest, principalmente por sus porcentajes superiores al 95%** en validación y casi el 95% en el testeo, si bien el algoritmo de vecinos más cercanos tiene mejores métricas, su tiempo de predicción es muy alto lo que lo hace ineficiente para un entorno de producción. No se elige 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 Random Forest, pero si el tema tiempo de respuesta es un factor clave se podría optar por usar la red neuronal básica).

6. Despliegue producción en modo API

Primeramente se construyó un archivo que cargue el modelo, y el PCA entrenados, que tome la librería del apartado 4.4.8 y al entregársele un audio, este archivo lo procese y lo envíe al modelo para cotejar una respuesta a la predicción del modelo.

Para el desarrollo de esta API se usó el framework de flask, el cual es una librería de Python para hacer APIs sencillas, mayormente para el backend, y por su misma sencillez se logra una velocidad en los servicios que se despliegan, además de que su código es sencillo y fácil de mantener, por supuesto el código se encuentra en el repo de este proyecto, esta API sencillamente la conectamos al archivo anterior que procesa el modelo.

La API se hizo de forma sencilla, con el objetivo de que fuera funcional, ya en un futuro se podría hacer un front, ya sea en react, angular, node o incluso boostrad para brindar la capa de presentación mejorada, ya con JWT y demás parámetros de seguridad y por supuesto que la API se despliegue por https para mejorar las vulnerabilidades.

Ya habiendo probado esta API, y viendo su funcionamiento, lo siguiente es hacerla portable y sostenible en el tiempo, para ello se usó docker y docker-compose, dentro del cual se empaqueto la API, con algunas pruebas y algunos ajustes por fin salió la versión 1.0, la cual se subió al repo de imágenes (dockerhub, 2022) y facilitar así su despliegue en cualquier servidor con un archivo de docker-compose como el siguiente:

```
version: "3.7"
services:
  app:
    image: wisrovi/tfm_2022:v1
    container_name: TFM2022
    ports:
      - 52000:2022
```

Ilustración 31. Contenido archivo despliegue con docker-compose

Fuente autor

Este archivo descargará la imagen ya completa, que ya contiene la API y las dependencias para funcionar y bajo el puerto 52000 expondría la salida a la API, por supuesto para lograr esta facilidad previamente se ha subido la imagen en la web, quedando así:

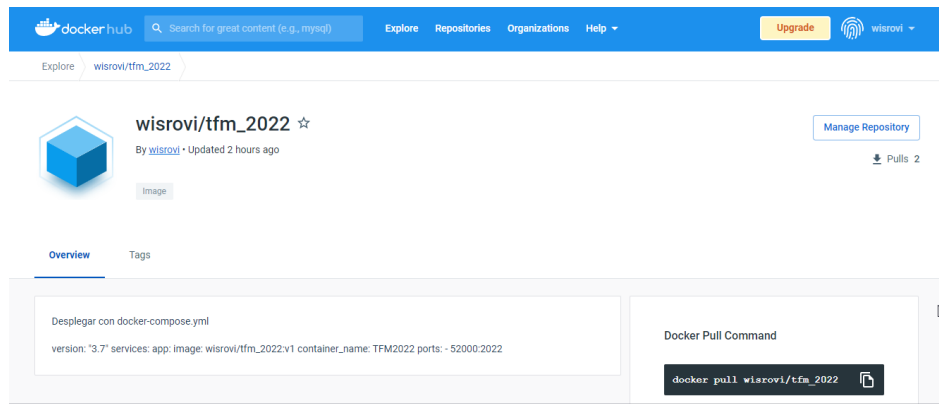


Ilustración 32. Imagen en dockerhub

Fuente autor (wisrovi, dockerhub, 2022)

Ya con esto desde un navegador se podrá acceder a la sencilla interfaz para usar el modelo, viéndose así desde el navegador:

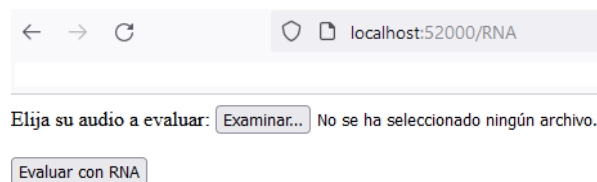


Ilustración 33. Usando la API

Fuente autor

Luego de usar la API, dando clic en examinar, seleccionar un archivo de audio wav en tiempo de dos segundos y sencillamente dar clic en Evaluar, esta devolverá la predicción y el tiempo que le tomo generar la predicción en formato json, así:

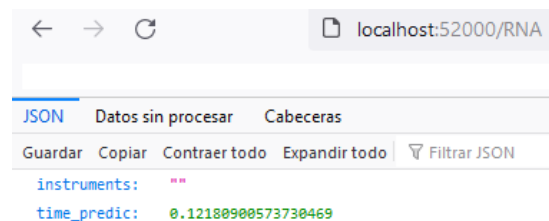


Ilustración 34. Respuesta de la API

Fuente autor

Y con esto ya solo es cuestión de probar con audios capturados desde un micro en tramos de 2 segundos e irlos enviando a la API.

7. Conclusiones y siguientes pasos

7.1. 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.

7.2. Sigüientes 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, 2022)

Bibliography

- Albaicín, S. d. (2014). *Espectograma*. Obtenido de <https://www.granada.org/inet/sonidos.nsf/d483b298c3f6a1b9c1257cdd00384c53/3fdcf36a7489b607c1257cde0024bb34!OpenDocument#:~:text=El%20espectrograma%20es%20una%20representaci%C3%B3n,representa%20en%20el%20eje%20horizontal.>
- analyticsvidhya*. (2022). Obtenido de <https://editor.analyticsvidhya.com/uploads/23385Capture6.PNG>
- Apren. Super.* (2022). Obtenido de <https://manualestutor.com/wp-content/uploads/Aprendizaje-supervisado-o-no-supervisado.jpg>
- ArbolDecision*. (2022). Obtenido de <https://oscarblancarteblog.com/wp-content/uploads/2014/08/tiposdenodos.png>
- bagging*. (2022). Obtenido de <https://machinelearningparatodos.com/cual-es-la-diferencia-entre-los-metodos-de-bagging-y-los-de-boosting/>
- Cañón., J. S. (2018). *Automatic Instrument Recognition with Deep Convolutional Neural Networks*. Obtenido de <https://www.bibliotecadigitaldebogota.gov.co/resources/2089794/>
- Chris Duxbury, M. D. (Dic de 2001). *Separation of Transient Information*. Obtenido de <https://d1wqtxts1xzle7.cloudfront.net/68684233/duxbury-with-cover-page-v2.pdf?Expires=1664420679&Signature=JvEER508QDN6cnGxKik8bRI4sHa8zb1D0q7OcvNvhodrQEKaEkbI2BnTNzfdQKDRPbiDj-sA2Or5LlsJIV57CvCV5SE3phltx4FCAvn6V8040yT487In0RuFkIP3QXbxdURmYxJ~O4mxDP686x4L>
- dockerhub*. (2022). Obtenido de <https://hub.docker.com/>
- Doshi, S. (2018). *Music Feature Extraction in python*. Retrieved from <https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d>
- Doshi, S. (30 de December de 2018). *Music Feature Extraction in Python*. Obtenido de <https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d>
- E. Fonseca, J. P. (2017). *In Proceedings of the 18th International Society for Music Information Retrieval Conference*. Obtenido de Suzhou, china: <https://annotator.freesound.org/fsd/>

- Ejemplo RandomForest.* (2022). Obtenido de <https://www.freecodecamp.org/news/content/images/2020/08/how-random-forest-classifier-work.PNG>
- Euronics. (2022). *potencia RMS*. Obtenido de [https://www.euronics.es/blog/rms/#:~:text=RMS%20\(Root%20Mean%20Squared\)%20o,calidad%20del%20sonido%20que%20escucharemos](https://www.euronics.es/blog/rms/#:~:text=RMS%20(Root%20Mean%20Squared)%20o,calidad%20del%20sonido%20que%20escucharemos)
- freesound, E. S. (2022). *Royalty Free Sound Effects*. Obtenido de Music Dataset: <https://www.epidemicsound.com/sound-effects/instruments/>
- Fu, Z., Lu, G., Ting, K. M., & Zhang, D. (10 de Dic de 2010). *A Survey of Audio-Based Music Classification and Annotation*. Obtenido de <https://ieeexplore.ieee.org/abstract/document/5664796>
- GOMEZ, S. J. (Julio de 2018). *GENERACIÓN Y EVALUACIÓN DE SECUENCIAS MELÓDICAS MEDIANTE INTELIGENCIA ARTIFICIAL*. Obtenido de https://oa.upm.es/53396/1/TFG_SILVIA_JIMENEZ_GOMEZ.pdf
- Heras, J. M. (18 de Septiembre de 2020). *Random Forest (Bosque Aleatorio)*. Obtenido de <https://www.iartificial.net/random-forest-bosque-aleatorio/>
- iberdrola. (2022). *¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?* Obtenido de <https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial>
- IBM. (2022). *¿Qué es Machine Learning?* Obtenido de <https://www.ibm.com/co-es/analytics/machine-learning>
- IBM. (2022). *Árboles de decisión* . Obtenido de <https://www.ibm.com/es-es/topics/decision-trees>
- interactivechaos.* (2022). Obtenido de https://interactivechaos.com/sites/default/files/inline-images/confusion_matrix.png
- javatpoint.* (2022). Obtenido de <https://static.javatpoint.com/tutorial/machine-learning/images/k-nearest-neighbor-algorithm-for-machine-learning2.png>
- John Thickstun, Z. H. (2017). *LEARNING FEATURES OF MUSIC FROM SCRATCH*. Obtenido de <https://arxiv.org/abs/1611.09827>
- José Pérez de Arce, F. G. (Enero de 2013). *Clasificación Sachs-Hornbostel de instrumentos musicales: una revisión y aplicación desde la perspectiva americana*. Obtenido de

https://www.scielo.cl/scielo.php?pid=S0716-27902013000100003&script=sci_arttext&lng=pt

joserzapata. (2022). Obtenido de https://joserzapata.github.io/courses/mineria-audio/extraccion_caracteristicas/2-Extraccion_Caracteristicas_73_0.png

kaggle. (2022). Obtenido de <https://www.kaggle.com/>

kaggle, & Musinet. (2020). Retrieved from <https://www.kaggle.com/datasets/imsparsh/musicnet-dataset>

librosa. (2022). *python package for music and audio analysis*. Obtenido de <https://librosa.org/doc/latest/index.html>

Ma, E. (2019). *nlpaug*. Obtenido de <https://nlpaug.readthedocs.io/en/latest/#>

Ma, E. (Sep de 2022). *nlpaug*. Obtenido de Augmenter that apply pitch adjustment operation to audio: <https://nlpaug.readthedocs.io/en/latest/augmenter/audio/pitch.html>

MANUEL ENRIQUE ALDANA SÁNCHEZ, J. S. (7 de Junio de 2013). *RECONOCIMIENTO DE PARTITURAS MUSICALES POR MEDIO DE VISIÓN ARTIFICIAL*. Obtenido de <https://repositoriousco.co/bitstream/123456789/931/1/TH%20IE%200188.pdf>

Méndez Hernández, J. S. (2020). *IDENTIFICACIÓN DE INSTRUMENTOS MUSICALES A PARTIR DEL*. Obtenido de <https://repository.javeriana.edu.co/handle/10554/57546>

MIT. (2022). *Lego, scratch*. Obtenido de <https://scratch.mit.edu/>

Na8. (10 de Julio de 2018). *Clasificar con K-Nearest-Neighbor ejemplo en Python*. Obtenido de <https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/>

netron. (2022). *netron*. Obtenido de web para graficar redes neuronales: <https://netron.app/>

pimages. (s.f.). Obtenido de <https://pimages.toolbox.com/wp-content/uploads/2022/04/11040522/46-4.png>

programmerclick. (2022). *scikit-learn Naive Bayes GaussianNB ejemplo*. Obtenido de <https://programmerclick.com/article/21391478006/>

Rodrigo, J. A. (Agosto de 2016). *Regresión logística simple y múltiple*. Obtenido de https://www.cienciadedatos.net/documentos/27_regresion_logistica_simple_y_multiple

- Rodrigo, J. A. (Junio de 2021). *Análisis de Componentes Principales*. Obtenido de https://www.cienciadedatos.net/documentos/35_principal_component_analysis
- Salgado Díaz del Río, A. (2019). *Reconocimiento automático de instrumentos mediante aprendizaje máquina*. Obtenido de https://biblus.us.es/bibing/proyectos/abreproy/92353/descargar_fichero/TFG-2353-SALGADO.pdf
- studio-22. (15 de May de 2019). *Ancho de banda*. Obtenido de [https://www.studio-22.com/blog/enciclopedia/ancho-de-banda#:~:text=Ancho%20de%20banda%20%2D%20Referido%20al,es%20mayor%20a%203%20dB\)](https://www.studio-22.com/blog/enciclopedia/ancho-de-banda#:~:text=Ancho%20de%20banda%20%2D%20Referido%20al,es%20mayor%20a%203%20dB))
- Tezcan, B. (9 de Junio de 2021). *Why Using a Dummy Classifier is a Smart Move*. Obtenido de <https://towardsdatascience.com/why-using-a-dummy-classifier-is-a-smart-move-4a55080e3549>
- tibco. (2022). Obtenido de https://www.tibco.com/sites/tibco/files/media_entity/2021-05/neutral-network-diagram.svg
- towardsdatascience. (s.f.). *Fragmento de audio ampliado para identificar los cruces por cero*. Recuperado el 26 de August de 2022, de towardsdatascience: https://miro.medium.com/max/1400/1*scDEE3LEYKh0YG7DkgBPOA.png
- wikipedia. (s.f.). *masa del sonido*. Obtenido de https://es.wikipedia.org/wiki/Masa_de_sonido
- wisrovi. (10 de 2022). *dockerhub*. Obtenido de tfm_2022: https://hub.docker.com/r/wisrovi/tfm_2022
- wisrovi. (10 de 2022). *ProcessAudio*. Obtenido de <https://pypi.org/project/ProcessAudio/>
- WISROVI. (2022). *Repositorio de este trabajo*. Obtenido de Github: <https://github.com/wisrovi/TFM2022>

PONER ANEXOS AL REPO DE GITHUB

PONER ANEXO DE LA LIBRERÍA DE PYTHON PARA EXTRACCION DE CARACTERISTICAS

ANEXO LIBRERÍA PYTHON AUDIO DATA INCREMENTATION

ANEXO DESPLIEGUE EN DOCKERHUB

ANEXO CON EL LINK PARA PROBAR LA API