Trabajo Fin de Grado
Ingeniería de Tecnologías de Telecomunicación

Reconocimiento automático de emociones en la música utilizando aprendizaje máquina

Autor: Ana Parra Galindo
Tutor: Francisco José Simois Tirado

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

Sevilla, 2021

Trabajo Fin de Grado
Ingeniería de Tecnologías de Telecomunicación

Reconocimiento automático de emociones en la música utilizando aprendizaje máquina

Autor:
Ana Parra Galindo

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Reconocimiento automático de emociones en la música utilizando aprendizaje máquina

Autor: Ana Parra Galindo

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2021

A mi familia y amigos

A mi tutor

Resumen

La Inteligencia Artificial, una tecnología muy actual, está respondiendo a necesidades y avances, como sucede en el mundo de la medicina y de la automoción. Existen proyectos y estudios para lograr la detección del cáncer a través del procesamiento de imágenes con el Deep Learning o conseguir llegar a la conducción autónoma de un vehículo. Trasladándonos a algo más cotidiano, junto a estas capacidades, la Inteligencia Artificial está siendo usada por plataformas de música para recomendar música o hacer listas de reproducción de forma automática y personalizada.

Con estas evidencias como respaldo, en el presente trabajo se quiere demostrar la facultad de un algoritmo de detectar algo tan subjetivo como son las emociones o sentimientos en la música. Requerirá de un entrenamiento de datos procesados y preparados, y un análisis de las distintas posibilidades de modelos para encontrar el más eficaz.

Gracias a la forma que tienen de funcionar los algoritmos de Machine Learning, no serán necesarias miles y miles de canciones para que el algoritmo pueda clasificarlas y relacionarlas con el sentimiento que despierta en las personas, si no que basándose en características comunes que él aprende en el proceso de entrenamiento hará esa clasificación como si de cualquiera de nosotros se tratara, y sabremos que emoción está asociada a un tipo de música.

Con el amplio abanico de posibilidades que presentan los modelos de Machine Learning y Deep Learning, se tomarán las Redes Neuronales como centro del trabajo. Serán necesarias varias combinaciones de las mismas y pruebas hasta dar con un modelo óptimo.

Abstract

Artificial Intelligence, a very current technology, is responding to needs and advances, as is happening in the world of medicine and the automotive industry. There are projects and studies to achieve cancer detection through image processing with Deep Learning or to achieve autonomous driving of a vehicle. Moving on to something more everyday, along with these capabilities, Artificial Intelligence is being used by music platforms to recommend music or make playlists in an automatic and personalised way.

With this evidence as support, this work aims to demonstrate the ability of an algorithm to detect something as subjective as emotions or feelings in music. This will require a training of processed and prepared data, and an analysis of the different possibilities of models to find the most effective one.

Thanks to the functioning of Machine Learning algorithms, thousands and thousands of songs will not be necessary for the algorithm to classify them and relate them to the feelings they arouse in people, but rather, based on common characteristics that it learns in the training process, it will classify them as if it were any of us, and we will know what emotion is associated with a type of music.

With the wide range of possibilities presented by Machine Learning and Deep Learning models, Neural Networks will be the focus of the work. Several combinations and tests will be necessary until an optimal model is found.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Me enseñaron que el camino del progreso no era rápido ni fácil.

- Marie Curie -

1.1 Motivación

La música y los sentimientos guardan una estrecha relación, solo hay que escuchar una canción que nos gusta mucho para que nos despierte un sentimiento e incluso puede devolvernos un recuerdo. Un ejemplo de esta relación podría verse en la pandemia que ha sacudido el mundo en el año 2020, y que ha sacado altavoces a los balcones para animar y transmitir alegría gracias a la música.

Esta relación entre música y sentimiento, sumado a la capacidad de un algoritmo de detectar emociones en un fragmento de canción, permiten a muchas plataformas agrupar música o recomendar la próxima canción relacionada con la última reproducida.

Como ejemplo de esto último se pueden poner a Spotify o Youtube, por ser las más usadas. Spotify hace listas de reproducción por género, el cual a su vez tiene relación con los sentimientos, listas por estado de ánimo o para un tipo de tarea, como conducir, leer, trabajar o limpiar. En el ejemplo para Youtube, tras poner a reproducir una canción ya tiene una recomendación relacionada con el estilo o el artista para cuando termine y proponértela.

Tanto la música como las emociones son dos imprescindibles en nuestra vida y se abre hueco en ellas la Inteligencia Artificial como en tantos otros campos. La capacidad que tienen los algoritmos de aprender gracias a las características sacadas de audios e interpretar emociones evocadas por una canción como si de un humano se tratara, podrá verse representado en este trabajo.

1.2 Objetivos y enfoque

La elaboración de este trabajo tiene el fin de encontrar un modelo de Red Neuronal capaz de clasificar por emociones un conjunto de canciones.

El comienzo será la elaboración de una base de datos, de la cual se extraerán las características usadas para entrenar y probar una red. Cada característica es de un tipo, pueden ser de tipo numérico o cadena de texto, por ello llevarán su correspondiente análisis y preparación, no todos los tipos de datos son válidos.

Es importante partir de una buena base de datos, bastante completa, con canciones que presenten matices muy concretos de ese estado de ánimo, pero que también haya algunos que sean un poco más parecidos a los de sus vecinos, con el fin de ajustar aún más el algoritmo. Para el entrenamiento esto puede reducir la incertidumbre a la hora de identificar y clasificar.

El proceso de modelado puede ser algo más tedioso. Como todo estudio, es imprescindible realizar pruebas y ajustes de todos los parámetros e hiperparámetros de los que consta el modelo. Aquí el modelo estará centrado en las redes neuronales, dónde se hará especial énfasis en el estudio del número de capas y en las neuronas por capa.

De entre las opciones de diferentes estructuras de redes neuronales se seleccionará el modelo más óptimo, se

mostrarán los resultados de las diferentes pruebas y se evaluarán los datos arrojados cómo conclusión del proceso.

1.3 Organización de la memoria

El presente trabajo contará con la estructura mostrada a continuación.

El primer capítulo, en el que nos encontramos, cuenta con una pequeña introducción que refleja el propósito y proceso del trabajo.

El segundo capítulo estará dedicado a la Inteligencia Artificial, el Machine Learning y el Deep Learning. Se explicará cada uno de estos términos y cómo se desarrolla un posible modelo de Machine Learning.

A continuación, el tercer capítulo, contendrá la exposición de la relación existente entre la música y las emociones desde diferentes perspectivas.

En el cuarto capítulo aparecerá la parte práctica, dónde el propósito es reconocer las emociones en la selección de canciones que forman la base de datos del presente trabajo.

El quinto capítulo albergará los resultados del anterior y nos llevará a unas conclusiones finales sobre la capacidad de un algoritmo de relacionar música y emociones, que aparecerá en el sexto y último capítulo.

Todo ello se verá acompañado de la bibliografía en la que se ha apoyado la documentación, una lista de figuras y otra de tablas, mostradas a lo largo de la memoria, y un Anexo de apoyo.

2 TRES CONCEPTOS PARA UNA IDEA DE FUTURO

Algunas personas denominan a esta tecnología inteligencia artificial, cuando en realidad lo que va a permitir es que aumente la nuestra propia.

- Gin Rometty, Presidenta y Directora ejecutiva de IBM-

2.1 Inteligencia Artificial, Aprendizaje Máquina y Aprendizaje Profundo

Casi a diario, y de forma indirecta, las personas hacen uso de la Inteligencia Artificial. Esta está presente en las redes sociales, en los buscadores, en los anuncios que nos muestran, en los asistentes de voz e incluso en los correos, filtrando los mensajes de spam. Acciones cotidianas, que han sido posibles o mejoradas gracias a la Inteligencia Artificial [1].

Estos algoritmos están presentes incluso en la sanidad. Actualmente son usados para asistentes virtuales permitiendo el incremento de la interacción médico-paciente, los chatbots como apoyo seguro de información médica o dispositivos de monitorización de asistencia sanitaria [2].

En el mundo de la música también está presente la Inteligencia Artificial. Su practicidad es innegable a la hora de recomendar canciones en plataformas como YouTube, asociándolas por género o artista. También en la creación, dentro de la plataforma de Spotify, de playlist personalizadas para cada usuario según las reproducciones más frecuentes, llegando a agrupar canciones por idiomas y estado de ánimo, además de por géneros.

Todos estos avances se han dado gracias a la Inteligencia Artificial, unida al Machine Learning y al Deep Learning. A continuación, se expondrá que son cada uno de estos aprendizajes que nos acompañaran de aquí en adelante en nuestra sociedad, debido a que son el futuro; y qué las hace diferentes entre sí. Se mostrará que tipos existen y cómo se llega a un modelo que pueda actuar por si sólo sin necesidad de intervención humana. Aparecerán también las redes neuronales, las protagonistas del presente trabajo.

2.1.1 ¿Qué es la Inteligencia Artificial o IA?

La Inteligencia Artificial es la simulación del comportamiento humano. Los algoritmos usados analizan el entorno, estudian cómo se comporta y generan una respuesta. Su aplicación y uso, como se ha comentado anteriormente, abarca casi todas las áreas dónde está presente la acción humana [3].

El concepto de Inteligencia Artificial se nombró por primera vez en 1956 en la Conferencia de Dartmouth, pero anteriormente ya se habían dado pinceladas sobre la misma. Descartes, en 1637, contempló la posibilidad de robots que pensasen, o Raimundo Lulio, entre los siglos XIII y XIV, escribió una obra dónde diseñaba una máquina lógica teórica para probar la verdad o falsedad de las afirmaciones o de los postulados [4].

A continuación se muestran los hitos más destacados de la historia de la Inteligencia Artificial, incluida su etapa más fría [5]:

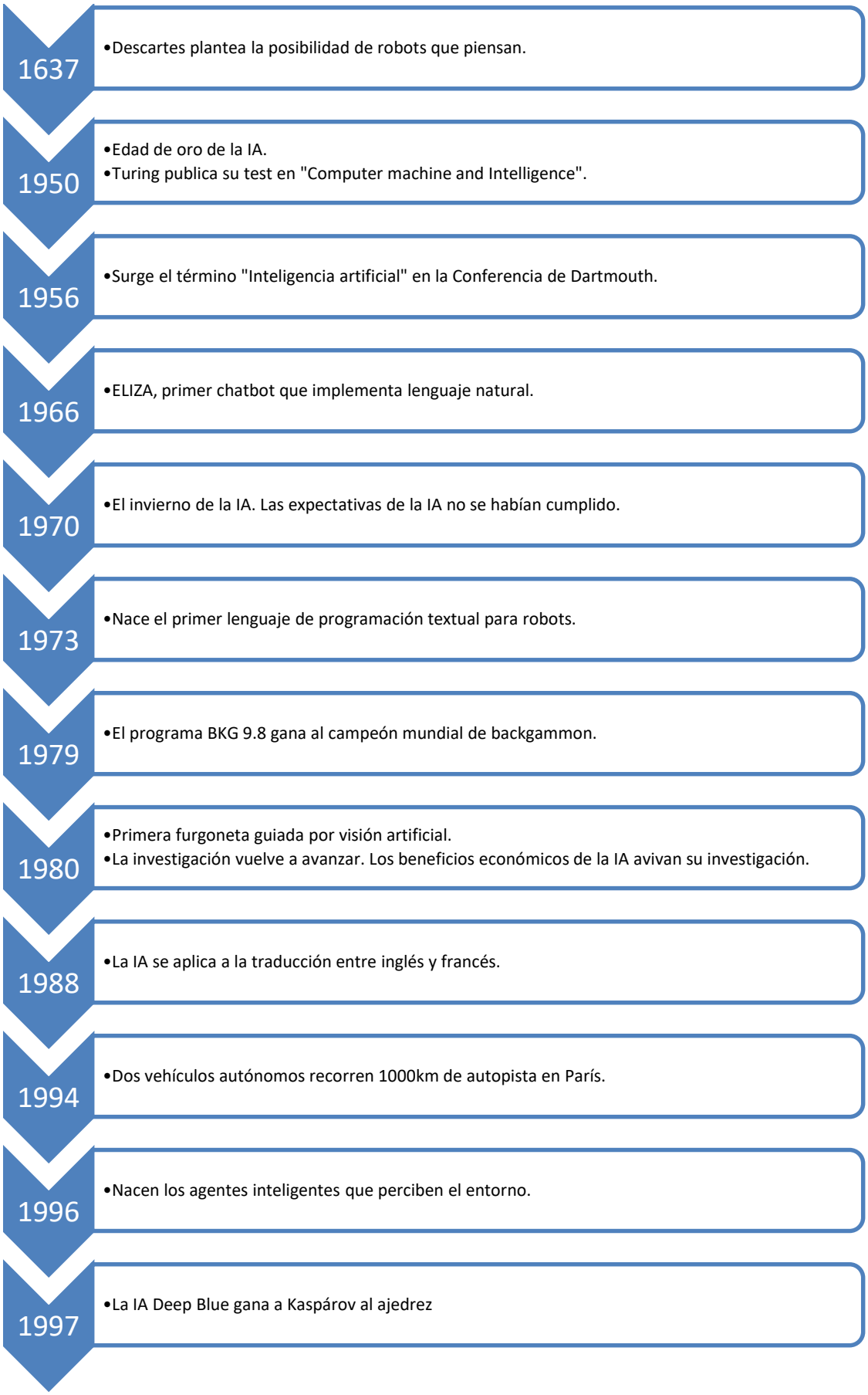

Figura 2-1. Cronología del Machine Learning desde su inicio hasta finales del siglo XX [5].

Con la entrada del nuevo siglo, el sigo XXI, los avances se daban de forma más consecutiva en el tiempo, reflejo del auge de la tecnología y de los datos. La entrada en el mundo del Big Data contribuyó al crecimiento de la IA, debido a que los datos son su alimento.

El Big Data hace referencia a un conjunto de datos que se caracteriza por su volumen, su velocidad y su variedad. Actualmente todo dispositivo conectado a la red genera datos útiles, por lo que hay un gran volumen; su renovación es rápida, todo en la red se desarrolla en un instante; y son variados, desde números hasta imágenes, pasando por vídeos o textos [6].

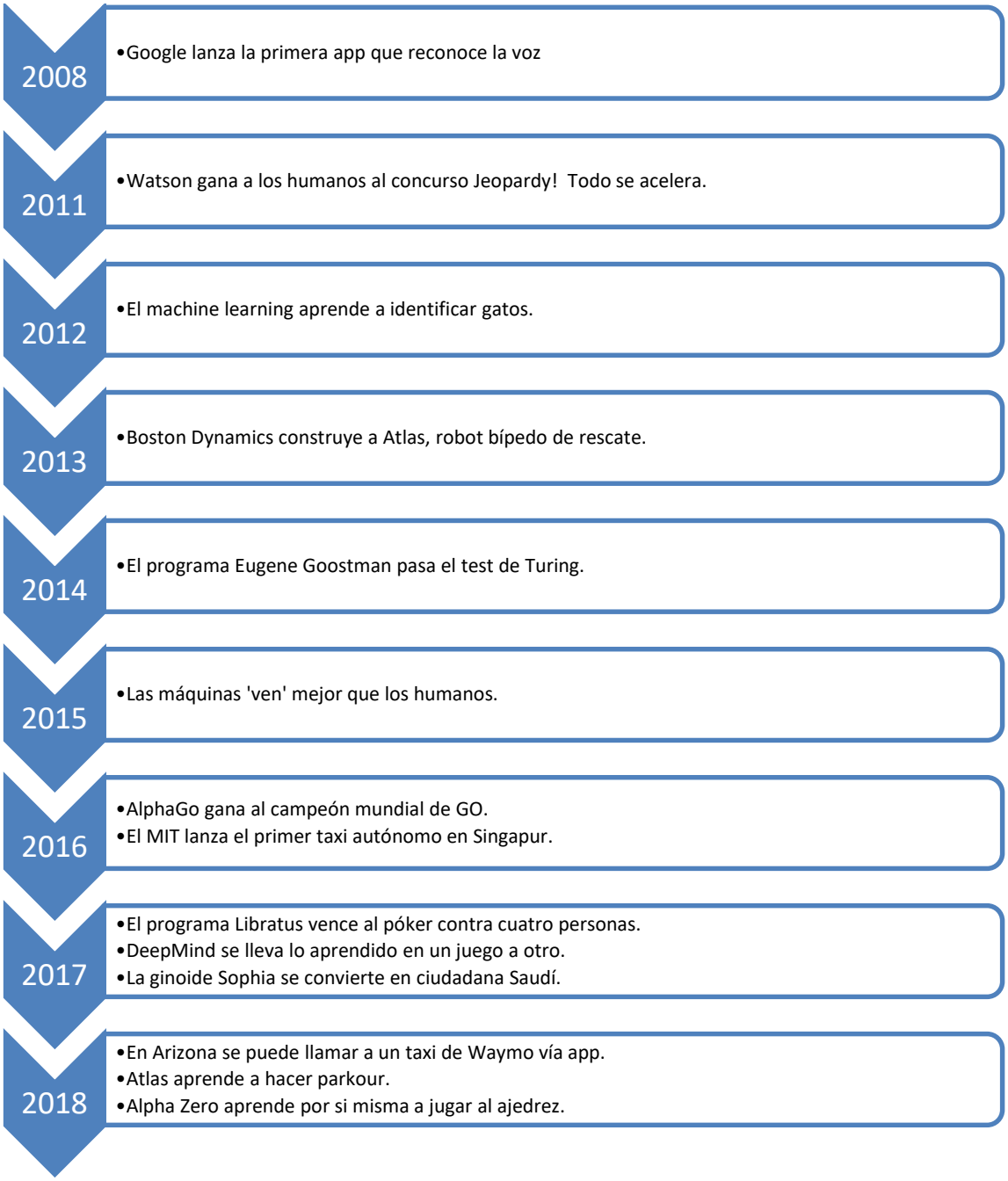

Figura 2-2. Cronología del Machine Learning sigo XXI [5].

Desde el 2018 hasta nuestros días han seguido sucediéndose acontecimientos gracias al Machine Learning, de hecho son tantos que se pisan unos a otros y cuesta destacarlos.

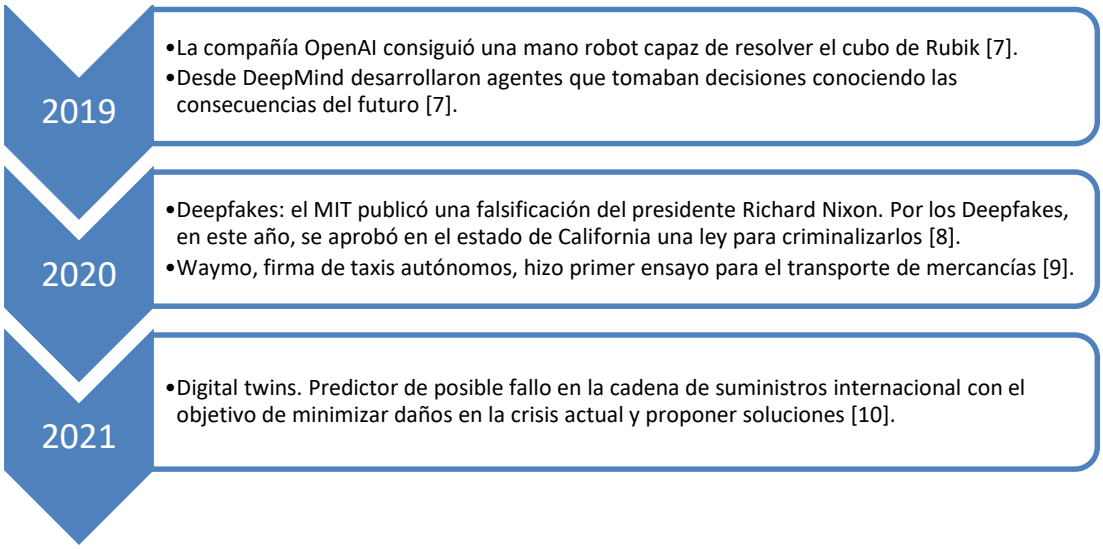

Figura 2-3. Cronología del Machine Learning sigo XXI más actual.

2.1.2 ¿Qué es el Machine Learning o Aprendizaje Automático?

Dentro de la Inteligencia Artificial existe un subconjunto denominado Machine Learning. Esta técnica de análisis de datos implementa métodos de cálculo en los algoritmos para que puedan aprender directamente de las entradas de datos, mejorando el rendimiento de forma adaptativa. No hacen uso de una ecuación predeterminada como modelo, encuentran patrones que les ayuda a tomar decisiones y hacer predicciones [11].

El algoritmo aprende a base de ejemplos, denominados conjunto de entrenamiento. Cada ejemplo es una instancia o muestra, de entrenamiento. Con cada nueva instancia el algoritmo se modifica y se ajusta, para predecir futuros escenarios y tomar decisiones. La medida de rendimiento es la precisión [12].

Tom Mitchell (1997) decía que un programa informático aprende de la experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.

La eficacia guarda relación con la cantidad de información recibida. Cuantos más datos diferentes reciba durante el entrenamiento, mayor complejidad tendrá el cálculo de la salida y más preciso será el proceso de elección. Para llegar a un algoritmo eficaz, se necesita un mínimo de datos con relevancia, que dependerá de la aplicación [12].

De forma generalizada, el aprendizaje automático es útil para aquellos problemas dónde la solución requerida necesite de muchas reglas o criterios, o aquellos casos que sean muy complejos y resulte costoso dar con la solución. También pueden ser interesantes en los casos con una gran cantidad de datos para trabajar, o dónde además, se actualicen con rapidez.

2.1.3 ¿Qué es el Deep Learning o Aprendizaje Profundo?

El Deep Learning es una forma especializada de aprendizaje automático. Es una técnica de Machine Learning que aumenta la precisión, consiguiendo resultados que no se habían visto antes, incluso superando en ocasiones el rendimiento humano.

El modelo se entrena con grandes cantidades de datos etiquetados, que pueden ser imágenes, texto o sonido. El propio modelo extrae las características directamente de estos datos, no necesita una selección previa. Esta gran cantidad de datos, junto con una alta potencia de cálculo, son los requisitos básicos para poder hacer uso del Deep Learning.

El Deep Learning o Aprendizaje Profundo también es conocido como redes neuronales profundas, debido al alto uso de arquitecturas de redes neuronales en los modelos, de las que se hablará más adelante en este trabajo [13].

2.1.4 Diferencias entre Machine Learning y Deep Learning

Cómo se ha comentado, el Aprendizaje Profundo es una rama del Aprendizaje Automático y sus modelos son mucho más superiores en los escenarios dónde podría hacerse uso de ambos. Las principales diferencias entre ambos aprendizajes se muestran en la siguiente tabla [14].

Tabla 2-1. Comparativa entre el Machine Learning y el Deep Learning [14]

	MACHINE LEARNING	DEEP LEARNING
Tiempo de ejecución (entrenamiento + prueba)	Modelos más simples, menos tiempo.	Modelos más complejos, más tiempo.
Tamaño de datos	En función de la aplicación.	Gran tamaño.
Extracción de características	Manual, previo al modelo.	Directamente de los datos.
Número de parámetros	Menor número de parámetros aprendibles e hiperparámetros.	Mayor número de parámetros aprendibles e hiperparámetros.
Relación entre rendimiento y cantidad de datos	Directamente proporcional hasta un punto, dónde empieza a ser inversamente proporcional.	Directamente proporcional. A más datos, más robusto es el modelo.
Capacidad de procesamiento (GPU)	Bajo rendimiento.	Alto rendimiento.
Capacidad de aprendizaje	Menor ajuste y sobreajuste.	Mayor ajuste y sobreajuste.
Interpretabilidad	Alta.	Baja.

2.2 Tipos de Sistemas de Aprendizaje Automático

Los criterios que se siguen para esta clasificación son la supervisión humana, la capacidad de aprender sobre la marcha de forma incremental, o si presentan la capacidad de detectar patrones en los datos de entrenamiento o solo aprenden comparando con datos conocidos.

A continuación, se muestran las diferencias y las características de las tres clasificaciones de aprendizaje que presentan los algoritmos [15].

2.2.1 Aprendizaje supervisado, no supervisado, semisupervisado o por refuerzo

- Aprendizaje supervisado: dentro de sus capacidades se encuentran la de realizar una clasificación o predecir un valor. Esta última también es conocida como regresión. Lo interesante es que los algoritmos de clasificación pueden usarse en regresión y viceversa.
  Los datos de entrenamiento que utiliza llevan etiquetas o clases identificativas. Además de esto, les acompañan a los ejemplos predictores o características.

  Algoritmos:

  - K-vecinos cercanos.
  - Regresión lineal.

- Regresión logística.
- Máquinas de vectores de apoyo (SVMs).
- Bosques de decisión o bosques aleatorios.
- Redes neuronales (RNA).

- Aprendizaje no supervisado: la diferencia con los supervisados está en la utilización de datos de entrenamiento sin etiquetas o marcadores.
  Entre las tareas que se pueden llevar a cabo con este algoritmo se encuentran reducir la dimensión de los datos (se simplifica la información encontrando la correlación entre dos características, de las que se extrae una nueva que englobaría a las anteriores), detectar anomalías, casos diferentes; o encontrar patrones que permitan relacionar datos que originalmente parecían inconexos.

  Algoritmos:

  - Agrupamiento o clasificación.
    - ✓ K-Medias.
    - ✓ DBSCAN.
    - ✓ Análisis jerárquico de conglomerados (HCA).
  - Detección de anomalías y de novedades.
    - ✓ SVM de una clase.
    - ✓ Bosque de aislamiento.
  - Visualización y reducción de la dimensionalidad.
    - ✓ Análisis de componentes principales (PCA).
    - ✓ Núcleo PCA.
    - ✓ Incrustación local del revestimiento (LLE).
    - ✓ T-Incrustación distribuida de vecinos estocásticos (t-SNE).
  - Aprendizaje de reglas de asociación.
    - ✓ A priori.
    - ✓ Brillar.
- Aprendizaje semisupervisado: los datos de entrada son una mezcla de los dos casos anteriores, hay datos marcados con etiquetas y otros que no llevan ninguna. Por ello, el algoritmo usado en estos casos es también una combinación de los supervisados y los no supervisados.
  La parte no supervisada es capaz de juntar los datos en grupos según las características que muestran, sin el uso de etiquetas; y en la parte supervisada entraría el etiquetado de los grupos.
- Aprendizaje por refuerzo: en este tipo de algoritmo el sistema de aprendizaje se denomina "agente". Este agente es el encargado de estudiar el entorno y generar una "acción" sobre él. Esta acción recibirá una respuesta, positiva o negativa, llamada "recompensa"; y según esa recompensa, el agente aprende.
  La finalidad del agente es la de encontrar una estrategia, llamada "política", para obtener la mayor recompensa.

2.2.2 Aprendizaje por lotes o en línea.

- Aprendizaje por lotes: el algoritmo no puede aprender de forma incremental. Su funcionamiento parte del conjunto de datos del que se dispone, y se entrena con ellos. Una vez haya aprendido el algoritmo se lanza a producción. Si posteriormente se necesitan introducir más datos, el proceso comienza desde el inicio, pero ahora el conjunto de datos aumenta, porque incluye tanto los datos iniciales como los nuevos.
  Este tipo de algoritmo no es adecuado para casos en los que haya una gran cantidad de datos o los datos crezcan muy deprisa.
- Aprendizaje en línea: a diferencia del caso anterior, este algoritmo si es capaz de aprender de forma incremental, según van llegando nuevos datos. A medida que entran nuevos datos, el

sistema aprende sobre ellos, y descarta los antiguos. Si la Tasa de Aprendizaje es alta, el proceso es rápido, afectando tanto al aprendizaje como al descarte de datos.
Este algoritmo es útil para sistemas con grandes cantidades de datos o con capacidad limitada.

2.2.3 Aprendizaje basado en instancias o basado en modelos.

- Aprendizaje basado en instancias: el algoritmo aprende de memoria. A esta memoria se le puede sumar una medida de similitud, para que además de contemplar en la clasificación aquellos casos que son idénticos a los aprendidos, también se puedan clasificar los que cumplan dicha medida. La medida de similitud permite al algoritmo generalizar los casos.
- Aprendizaje basado en modelos: para generalizar lo aprendido a otros casos nuevos, se hace uso de un modelo. Este modelo nace del estudio de los datos de entrenamiento, que permitirá realizar las predicciones de los nuevos datos.
  Para comprobar el rendimiento o buen funcionamiento del algoritmo se hace uso de una medida. Esta medida puede ser una función de utilidad que mida lo bueno que es un modelo, o una función de coste que mida lo malo que es.

2.3 Redes Neuronales

También llamadas redes neuronales artificiales o RNAs. Son un modelo de computación cuya estructura se inspira en las redes de neuronas presentes en el cerebro humano. Este modelo de computación es el núcleo del Aprendizaje Profundo. Aprenden de los datos para identificar patrones, clasificar o predecir eventos futuros [15] [16].

2.3.1 El perceptrón o RNA

La unidad más pequeña de RNA es el perceptrón, presentada en 1957 por Frank Rosenblatt, análogo a una neurona humana. Posee entradas, que podrían asociarse a los impulsos que recibe una neurona en el cerebro. Estas entradas tienen unos pesos, que indicarían la intensidad de la sinapsis (relación entre dos neuronas cerebrales). Cada una de estas entradas se conecta con la TLU, la unidad lógica de umbral, encargada de sumar las entradas ponderadas por sus pesos respectivos, y aplicar la función de activación, que dará lugar a la salida. La salida del perceptrón es la respuesta al estímulo, y se transmite a la próxima neurona, simulando ser un axón.

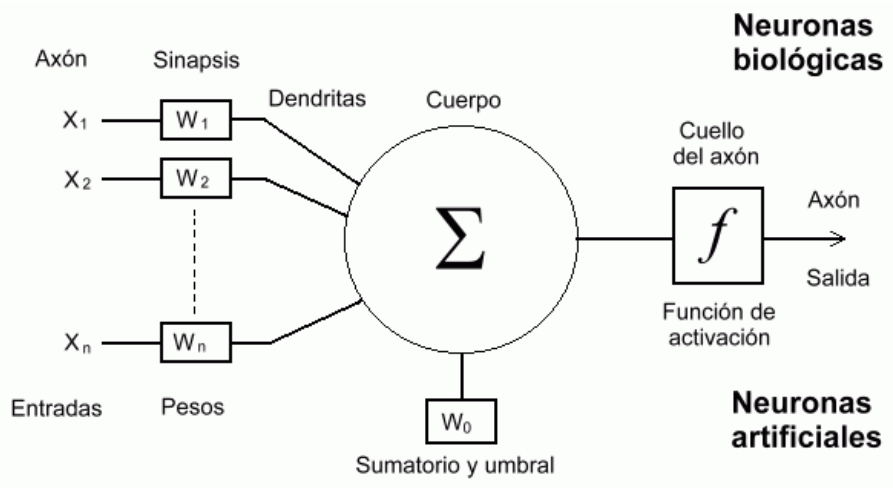

Figura 2-4. Similitud entre una neurona biológica y un perceptrón [17].

La TLU calcula una suma ponderada de las posibles entradas, conocida como función de red, a la que posteriormente se le aplica una función de paso. Los pesos son los que marcan el potencial de cada entrada. La

función de paso o de activación más común es la función escalonada de Heaviside o la función signo [18].

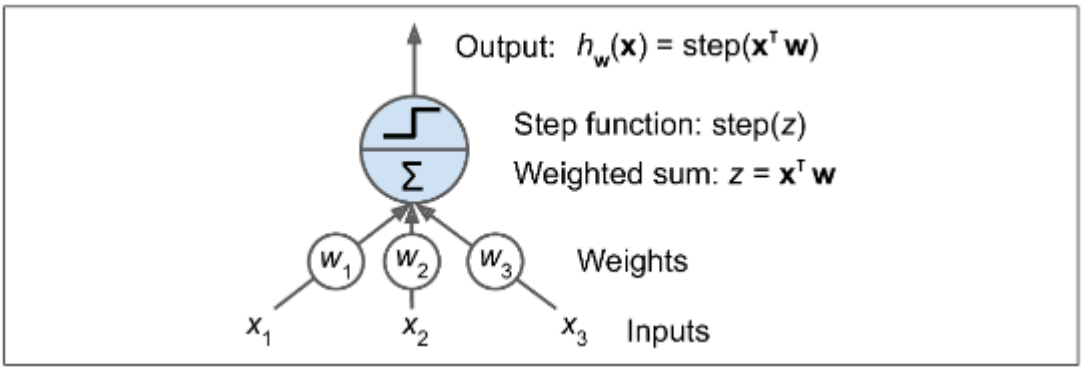

Figura 2-5. Unidad básica de neurona artificial [15].

2.3.2 Redes Neuronales

Los perceptrones no suelen ir aislados, se unen varios para formar las conocidas redes neuronales. Estas redes, de forma básica, constan de dos capas, la capa de entrada con una neurona de sesgo inicializada a uno; y la capa de salida, que es la que arroja los resultados de la red.

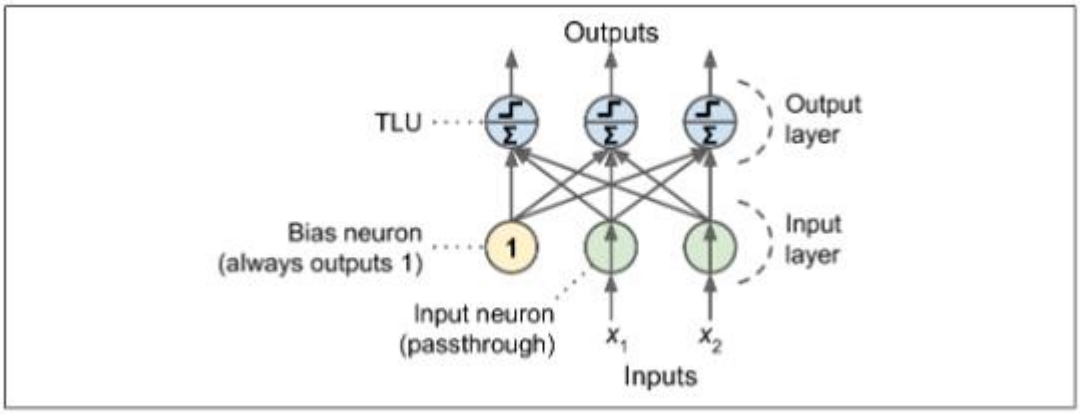

Figura 2-6. Arquitectura de un perceptrón con dos entradas y tres salidas [15].

Las conexiones entre neuronas de diferentes capas puede ser total, todas las neuronas de una capa se conectan con todas las neuronas de la capa siguiente, y en ese caso se denomina capa densa. Toda conexión dentro de la red siempre lleva asociado un peso.

Si la capa de salida cuenta con más de una neurona de salida, se denomina red multietiqueta, permitiendo hacer clasificaciones o predicciones para diferentes clases.

El entrenamiento de un perceptrón sigue la regla de Hebb con el objetivo de ajustar los pesos que identifican a cada una de las conexiones. Esta regla hace uso del error cometido en una predicción errónea, con él se ajustan los pesos de las conexiones de entrada que contribuyeron al error.

Entre las limitaciones que presentan las redes anteriores se encuentra la incapacidad de aprender patrones complejos, debido a que las salidas son lineales, y no pueden calcular probabilidades, solo predicen de acuerdo a un umbral sólido. Algunas de estas limitaciones se eliminan sumando más perceptrones. La acumulación de estos daría paso a las RNM, Perceptrón Multicapa [15].

2.3.3 Redes Multicapa o RNM

La arquitectura está construida por una capa de entrada, varias capas ocultas, y una capa de salida. Todas las capas, exceptuando la de salida, tienen una neurona de sesgo conectada a todas las TLUs de la capa siguiente.

El entrenamiento de estas redes multicapa sigue el algoritmo de retropropagación, que al igual que el entrenamiento del perceptrón, se basa en el error cometido por la red en la predicción, en este caso, calcula el gradiente del error. Con esto, el objetivo vuelve a ser el ajuste de los pesos de cada conexión, añadiendo el cálculo del valor correcto del sesgo.

Para poder llegar a calcular un gradiente de error la función de paso de las neuronas no puede ser la función escalón, necesita una función que presente derivada no nula. Dentro de las posibilidades de funciones de paso con derivadas no nula estaría la función logística, la función tangente hiperbólica o la función de unidad lineal rectificada.

El algoritmo necesita pasar dos veces por la red para obtener el gradiente de error. La primera pasada es desde la capa de entrada hacia la capa de salida, haciendo una primera predicción, valiéndose de un pequeño lote de datos. Conseguida esta primera ejecución, se calcula el error cometido, y empleando la regla de la cadena, averigua que contribución hacen cada una de las conexiones de la red al error. El segundo cálculo se realiza transcurriendo de nuevo por la red, desde la capa de salida a la capa de entrada. Este paso inverso permite obtener el gradiente de error gracias a los pesos. Con este último paso se obtiene el gradiente del error de la red. Lo que quedaría es cambiar los pesos para minimizar el error del próximo cálculo.

El inicio de los pesos debe ser aleatorio para que el entrenamiento no se vea condicionado [15].

Estas redes se pueden profundizar todo lo que se quiera, sumando capas y neuronas consecutivamente.

2.4 Construir un modelo de Machine Learning de forma generalizado

1. Estudiar el problema.

   El primer paso es un análisis del problema y marcar el objetivo al que se quiere llegar. Concretar esta información podría ayudar a la elección del modelo y medida del rendimiento.

2. Obtener los datos.

   Los datos pueden venir por la misma persona que solicita resultados, por bases de datos a las que tengamos acceso o haciendo uso del web scrapping, recopilando información de internet y creando desde cero una base de datos.

   Hay que tener presente que es importante la calidad y la cantidad de los datos, esto repercutirá en el modelo [19].

3. Analizar y visualizar los datos.

   Del análisis, representación gráfica y pruebas de los datos, se pueden obtener patrones o relaciones entre diferentes características que a simple vista no se aprecian, se pueden localizar atributos discriminatorios entre clases o atributos que no aportan información.

   Esta información será útil para el paso siguiente dónde se prepararán los datos para usarlos como instancias de entrenamiento.

4. Preparar los datos.

   Para conseguir un conjunto de datos válidos para el modelo es necesario tratar los siguientes datos que se puedan encontrar:

   - Reducción de dimensiones.

     Entre los datos recopilados puede existir redundancia o características no discriminatorias, por ello se reducen las características, lo que conlleva una reducción de la dimensión.

     Una forma de disminuir es eliminar características poco importantes que se hayan descubierto en el paso de análisis y visualización de datos, siempre que se esté seguro de su baja influencia.

Otra forma es crear nuevas características combinando iniciales, consiguiendo que las nuevas sean independientes entre ellas [20].

- Eliminar duplicados.

  La existencia de datos de entrenamiento duplicados podría llevar al modelo a error.

- Datos balanceados.

  Este tipo de datos se da en los problemas de clasificación donde tenemos en los datos de entrenamiento clases con representación mayoritaria y clases con representación minoritaria. Este desequilibrio puede provocar que el modelo no generalice de forma adecuada y se equivoque en la elección de la clase final, aun dando una buena medida de error. Esto perjudica a las clases minoritarias [21].

  Para tratar esta cuestión se recomienda primero entrenar el modelo con los datos originales aunque se encuentren desbalanceados. Si el modelo no presenta ningún error, los datos se dan por válidos. En el caso contrario, se pueden tomar dos caminos. El primer camino es aplicar "undersamplig" y "oversampling". Con el "undersampling" conseguimos disminuir las muestras de la clase mayoritaria de forma aleatoria; y con el "oversampling", conseguimos lo opuesto en las clases minoritarias haciendo uso de muestras sintéticas. El segundo camino es la aplicación de pesos en la función de coste, con estos pesos se penaliza a la clase mayoritaria, y se beneficia a la clase minoritaria [22].

- Normalizar los datos.

  Los datos no escalados o normalizados pueden hacer que se dé más importancia a unos valores que a otros según su escala, con independencia de la característica.

  Existen dos formas para acometerlo, el escalado o la normalización. El escalado va a permitir que los valores de las características estén dentro de un mismo rango, comúnmente se elige [-1, 1] o [0, 1]. Este escalado también se conoce como la normalización MinMax, asigna el límite inferior del rango al valor más pequeño de la característica, y el límite superior al más grande, los demás valores van escalados entre ambos.

  La forma estándar de la normalización permite que todos los valores de las características tengan el mismo valor de media y desviación típica; típicamente se usa varianza 1 [23].

- Corregir valores erróneos.

  Estos valores son los campos en blanco o cadenas de texto. La corrección de los valores en blanco puede ser la supresión de los mismos, pero solo se recomienda cuando son una proporción muy pequeña. Otra práctica es rellenar con el valor 0, pero puede resultar engañoso [24]. Se puede recurrir a completarlos con los valores más repetidos, con la media, la mediana o la moda.

  Para no hacer uso de cadenas de texto como valores de características, se haría un cambio de la cadena por un número, guardando siempre la misma relación al hacer el cambio [25].

5. Modelado.

   Para este paso se seleccionan varios algoritmos de aprendizaje automático hasta dar con los que den mejores resultados, haciendo uso del conjunto de entrenamiento. Los resultados pueden dar a entender que la predicción es correcta, pero la comprobación del rendimiento es la que aclarará si el algoritmo es bueno o malo, no todos serán adecuados para el caso de estudio.

   Tras varias pruebas se llegará a un listado de posibles algoritmos para la resolución del problema. Estos serán los que mejor se ajusten al objetivo marcado al inicio del proceso y arrojen los mejores resultados de predicción.

6. Entrenamiento.

   Elegido el modelo más adecuado, se entrenará con los datos del conjunto de entrenamiento. Antes de iniciarlo, se debe asignar una combinación inicial de valores a los hiperparámetros del modelo, que pueden inicializarse de forma aleatoria [15]. Los hiperparámetros son aquellos parámetros que se

emplean para ajustar el entrenamiento del modelo [26].

Antes de pasar a la evaluación se puede hacer uso de la validación cruzada para asegurar el correcto funcionamiento del modelo. Para ello, del mismo conjunto de entrenamiento se extrae una sección de datos, denominados datos de validación, que permitirán predecir el comportamiento del modelo con un hipotético conjunto de datos de testeo [27].

7. Evaluación.

   Tomando el conjunto de datos más pequeño, el llamado conjunto de test, se evalúa el modelo entrenado. Se calculan las métricas y se determina la validez [25].

   La precisión del resultado del modelo debe superior al 50%, el uso del Machine Learning no puede equipararse al azar entre la cara o la cruz al lanzar una moneda al aire [25]. Si los resultado no son los deseados podría asociarse a un sobreajuste (el modelo se ajusta en exceso a los datos de entrenamiento y no generaliza) o subajuste (el modelo no es capaz de ajustar ninguna instancia). La solución es probar cambios en los valores de los hiperparámetros hasta conseguir aumentar la precisión del modelo [22].

   Si el cambio de estos parámetros no diese resultado se podría ir repasando todos los cambios y ajustes de pasos anteriores hasta dar con la solución, incluidos posibles cambios en los datos de inicio.

8. Inferencia.

   El último paso supondría ponerlo en marcha, supervisarlo y mantener el sistema. Exponer la conclusión debe llevar consigo mostrar lo aprendido, los pasos que se han dado, los fallos encontrados, las suposiciones realizadas y las conclusiones a las que se ha llegado.

3 LA MÚSICA Y LAS EMOCIONES

Sin sentimiento, todo lo que hagas equivaldrá a nada.

- Billie Holiday -

3.1 Visión biológica y fisiológica

Las reacciones que consigue la escucha de una melodía se traducen en alteraciones fisiológicas y biológicas. Alteraciones hormonales que actúan sobre el sistema nervioso, alteraciones del ritmo cardiaco… Y tras la escucha entran en juego el pensamiento y las emociones.

Para estudiar la relación de la música con la conducta humana existe una rama de la Psicología, conocida como Psicología de la Música, que intenta comprender el efecto que causa la música sobre las personas. El estudio se centra en dos dimensiones, la dimensión psicofisiológica y la dimensión psicobiológica.

Desde el punto de vista psicofisiológico, la música estimula tres partes del cerebro, la zona bulbar, el diencéfalo y el nivel cortical. Cada una de las partes da una respuesta diferente. La zona bulbar, centro de las reacciones físicas, guarda relación con el ritmo. El diencéfalo origina las emociones y los sentimientos, contenido básico de la conciencia y de la actividad psíquica, tras la recepción de la afectividad de la música. Por último, el nivel cortical, centro de la actividad intelectual y el pensamiento, responde con estímulos ante la armonía.

Desde el punto de vista de la psicobiología, la relación existente entre la música y el pensamiento musical viene determinada por la funcionalidad de las estructuras cerebrales. El cerebro involucra no solo el pensamiento, localizado en la zona cortical, si no también involucra la afectividad, localizada en otra parte diferente del cerebro, en el diencéfalo y en el lóbulo temporal. Esta última, es la característica básica del estado de conciencia [28].

Por ello podemos hablar de dos partes o procesos dentro del cerebro, la parte objetiva y la parte afectiva. La primera parte es llamada “Unidad de valoración objetiva” y es la llevada a cabo por las funciones del pensamiento. La segunda, es conocida por “Unidad de valoración afectiva”, es el resultado de la respuesta emocional del sistema límbico, el tálamo, el hipotálamo y el lóbulo temporal. La fusión de ambos da como resultado la conciencia.

Todo este estudio tiene presente que la cultura, el entorno y la inserción en el medio social de cada persona juegan un papel importante en las reacciones de dos individuos ante la misma melodía. Esto tiene un reflejo en lo biológico, no existen dos cerebros iguales, no existen dos redes de neuronas iguales. Cada persona desde que nace y según los estímulos que recibe a lo largo de su vida, va tejiendo las conexiones de sus neuronas de una forma distinta a las de otra persona [28].

3.2 Relación bidireccional

La relación de las emociones o estados de ánimo, y la música, es bidireccional. La música y los sonidos provocan en nosotros una respuesta según lo escuchado. Y según nuestro estado de ánimo o momento en el que nos encontremos, elegimos un tipo de música u otro.

Para ejemplificar lo comentado se puede observar el uso de una banda sonora de tensión o suspense para

provocar miedo en el espectador de una película de terror, o la necesidad de escuchar música rítmica que te invite a bailar cuando estás en una fiesta o celebración.

En Noruega se hizo un estudio para poner de manifiesto la relación bidireccional comentada. Para ello se valieron de la subjetividad de un grupo de jóvenes desconocidos por el investigador, se buscaba la mayor heterogeneidad posible, y se siguieron las siguientes fases [29].

Fase1: Un primer grupo de jóvenes iban a establecer la relación entre estado de ánimo-música. Se les mostraba a cada uno de ellos ocho emociones, las mismas para todos, y a cada emoción le tenían que asignar una canción que le hubiese suscitado esa emoción. De esta primera fase se obtiene una primera matriz de datos subjetivos, dónde aparecen ocho conjuntos de canciones asociado cada conjunto a una emoción.

La selección de las 8 emociones se hizo de acuerdo al diferencial semántico, y eran felicidad, relajante, animada, romántica, nostálgica, bailable, triste y tensión.

Fase 2: Se analizan los rasgos musicales de las canciones de la fase anterior. Se hace una selección de canciones típicas, ocho en total, una por emoción, y se obtiene una selección, que será la mostrada al grupo de validación. La selección de canciones típicas es la selección de las canciones más repetidas dentro de cada uno de los ocho grupos.

Fase 3: Un segundo grupo de jóvenes, diferente al primer grupo, iba a establecer la relación contraria, música-estado de ánimo. Su función era puntuar del 1 al 10 si esa canción despertaba en ellos la emoción con la que se había relacionado esa canción. De esta fase se obtiene otra matriz de datos subjetivos.

A la matriz obtenida en la fase 3, se le suma una tercera matriz de datos objetivos. Estos datos son los rasgos musicales de cada una de las canciones en examen, utilizando los más discriminatorios.

Tras el análisis de los datos recogidos durante el estudio desde la Universidad de Noriega llegaron a las siguientes conclusiones: “1) Existe una relación estadísticamente significativa entre la asociación de sentimientos a las canciones propuestas, primero “motu proprio”, y después validada externamente por otros sujetos entrevistados. 2) Existe dimensionalidad en la estructura de valoración de las canciones, que corresponde con las propuestas de dimensionalidad en la orientación psicológica del diferencial semántico. 3) Existe una relación empírica estadísticamente significativa entre las cualidades musicales de las canciones y su efecto psicológico tal y como lo reconocen los entrevistados." [29].

3.3 Tipos de emociones y representación en la psicología

Las emociones que se asocian a la música pueden ser de forma general de dos tipos, emociones inducidas o emociones percibidas, entendidas también como emociones en la música o emociones de la música.

La emoción inducida es la emoción buscada por el creador, el sentimiento que llevó a crear o componer esa canción u obra. Es una emoción más objetiva, pero aun así está sujeta a las circunstancias que rodean a la canción, y puede fluctuar con el tiempo. Un ejemplo de ello, un autor que tras muchas escuchas y ensayos, cansado, deja de percibir las emociones iniciales y le cambia la emoción a la canción.

La emoción percibida es la emoción experimentada por el oyente, el estado de ánimo que surge de la escucha de la canción o de la obra. Aunque esta emoción sea más subjetiva, las personas coinciden más en la emoción percibida que en la inducida.

Estudios psicológicos dieron lugar a dos posibles representaciones de las emociones. Estas serían la representación categórica o la representación dimensional [30].

La representación categórica divide las emociones en grupos o clústeres. Cada uno de ellos será identificado por un grupo de adjetivos. Uno de los modelos más recientes de representación categórica y que pone de manifiesto la teoría básica de la emoción, es el modelo de Juslin y Sloboda (2001) [31].

Tabla 3-1. Evaluaciones clave para emociones básicas adaptadas de Sloboda (2001) [30]

Emoción	Coyuntura del plan	Tema central de las relaciones
Felicidad	Subobjetivos alcanzados	Avanzar razonablemente hacia un objetivo
Ira	Plan activo frustrado	Una ofensa denigrante contra mí y los míos
Tristeza	Fracaso del plan principal o pérdida del objetivo activo	Habiendo sufrido una pérdida irremediable
Miedo	Objetivo de autoconservación amenazado o conflicto de objetivos	Enfrentarse a un peligro físico inmediato, concreto o abrumador
Disgusto	Golpe de efecto violado	Tomar o estar cerca de algo desagradable

Todo conjunto de emociones depende de cómo se definan las emociones, pero existe un consenso alrededor del modelo expuesto por Juslin y Sloboda. Las emociones básicas deben ser universales y son el conjunto más simple de emociones que pueden dar lugar a todas las demás, buscándole una similitud con los colores, las emociones básicas vendrían a ser los colores primarios.

La representación dimensional implica la distribución de las emociones a lo largo de dos ejes, un eje representa la excitación de la emoción (si la emoción es más activa o menos activa), y el otro eje representa la valencia (si la emoción es más positiva o más negativa) [30].

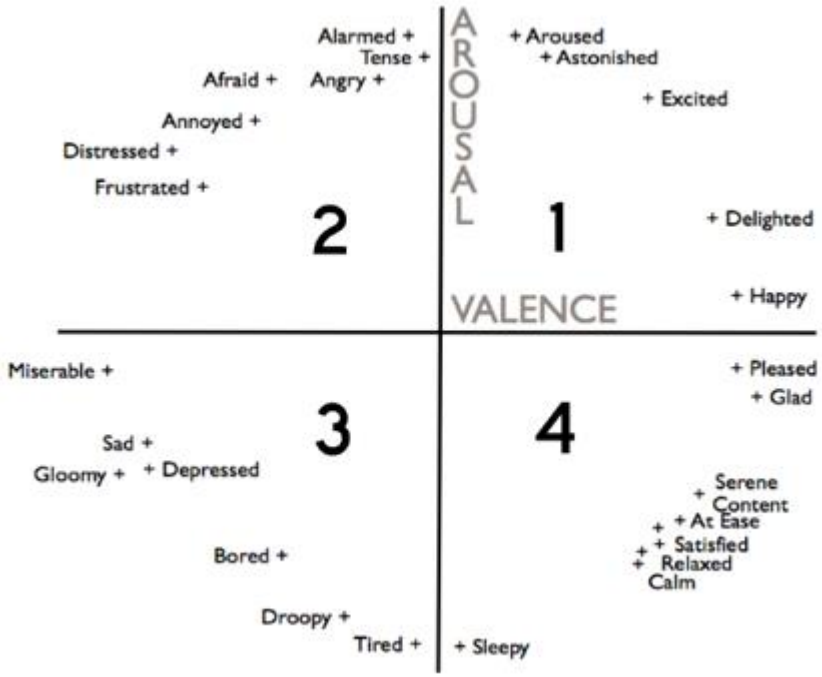

Figura 3-1. "Modelo circumplejo del afecto" con dimensiones de excitación y valencia, adaptado por Russell (1980) [30].

3.4 Elección de emociones

En el presente trabajo se van a detectar cuatro emociones básicas en la música. Estas cuatro emociones son

Alegría, Tristeza, Enfado y Relajado. La elección de solo estas cuatro emociones se apoya en el estudio y las conclusiones alcanzadas por Cyril Laurier en su tesis [30]. Es una referencia más avanzada resumida en este apartado para apoyar la elección de estas emociones.

Cyril Laurier ha querido validar si el etiquetado con estas cuatro emociones de canciones de una plataforma, cuadraba con los expertos. Los datos de los que partieron fueron las etiquetas sociales más repetidas. La matriz obtenida tenía una gran dimensionalidad, y para reducirla y mostrarla en un espacio de estado semántico, aplicaron Análisis Semántico Latente (LSA) con Descomposición del Valor Singular (SVD).

[El LSA es, en efecto, una técnica habitual para mapear palabras y documentos en un "espacio conceptual", mapeando la matriz de términos del documento (o, en nuestro caso, la matriz de etiquetas de las pistas) en un espacio conceptual reducido, que denominamos "espacio semántico del humor".]

Representación categórica.

El espacio semántico del estado de ánimo pasó a compararse con las representaciones propuestas por la literatura, con la representación categórica y con la representación de excitación/valencia.

Haciendo uso de un método de agrupación no supervisado a través del algoritmo de maximización de expectativas (EM) de Dempster et al. (1977), lograron una representación categórica del espacio semántico del que partió el estudio del departamento de la Universidad.

[El algoritmo EM no requiere una fase de entrenamiento. Consiste en un algoritmo iterativo, cuyo objetivo es encontrar los parámetros de la distribución de probabilidad que tiene la máxima verosimilitud de sus atributos. El primer paso es la Inicialización, seguida de la Expectativa y la Maximización, ambas repetidas iterativamente hasta la convergencia.]

Dicha representación categórica se denominó representación Folksonómica. Los cuatro clúster que surgieron son muy similares a la teoría básica de la emoción de Juslin y Sloboda (2001), mostrada en la Tabla 3-1 de la sección anterior; y a los cuatro cuadrantes de excitación-valencia de Russell, también mostrado en la Figura 3-1 de la sección anterior. Por tanto parece que no es muy desatinado seleccionar cuatro emociones como base de todas las demás.

No les bastaba con la similitud anterior, y calcularon la similitud Intra-clúster e Inter-Clúster de la representación folksonómica, junto con las de Hevner y MIREX, y las compararon.

Los clúster de Hevner y de MREX aparecen en las siguientes Tablas:

Tabla 3-2. Adjetivos de humor emparejados en el modelo de Hevner [30]

Clusters	Hevner Mood Adjectives
Cluster 1	Spiritual, sacred
Cluster 2	Pathetic, sad, mournful, tragic, melanholy, depressing, gloomy, heavy, dark
Cluster 3	Dreamy, tender, sentimental, longing, yearning, plaintive
Cluster 4	Lyrical, serene, tranquil, quiet, soothing
Cluster 5	Humorus, whimsical, playful, delicate, light
Cluster 6	Merry, joyous, gay, cheerful, bright
Cluster 7	Restless, passionate, exciting
Cluster 8	Martial, majestic

Tabla 3-3. Medios de similitud intra-clúster para cada taxonomía de estados de ánimo [30]

Clusters	MIREX Mood Adjectives
Cluster 1	Passionate, rousing, confident, boisterous, rowdy
Cluster 2	Rollicking, cheerful, fun, sweet, amiable, good natured
Cluster 3	Literate, poignant, wistful, bittersweet, autumnal, brooding
Cluster 4	Humorous, silly, campy, quirky, whimsical, witty, wry

Los resultados de las medidas de similitud entre los valores de un mismo clúster fueron los siguientes para cada una de las representaciones:

Tabla 3-4. Medios de similitud intra-clúster para cada taxonomía de estados de ánimo [30].

Mood Taxonomy	Intra-cluster similarity
Hevner	0.55
MIREX	0.73
Folksonomy	0.82

Los resultados de las medidas de similitud entre diferentes clústeres fueron los siguientes para cada una de las representaciones:

Tabla 3-5. Medias de disimilitud entre inter-clúster para cada taxonomía del estado de ánimo y su línea de base aleatoria para la comparación [30].

Mood Taxonomy	Inter-cluster dissimilarity
Hevner	0.70
MIREX	0.56
Folksonomy	0.9

Como puede observarse, la representación de MIREX es mejor en la similitud dentro del mismo clúster, sin embargo, no es tan clara la diferenciación entre unos y otros. En cuanto a la representación de Hevner, tiene mejor separación entre clúster. La representación Folksonómica arroja buenos resultados porque se ha extraído del propio espacio semántico de estudio.

Representación dimensional.

Para visualizar el espacio semántico del estado de ánimo en dos dimensiones, y poderlo comparar con la representación de Russel, hacen uso del algoritmo del mapa autoorganizado (SOM).

[Esta técnica de transformación de datos reduce la dimensión de un espacio de entrada mediante el uso de redes neuronales autoorganizadas. Una red neuronal artificial se entrena mediante aprendizaje no supervisado para construir una representación de baja dimensión de las muestras del espacio de entrada (normalmente entre 1 y 3 dimensiones para permitir una representación visual).]

El resultado aparece en la Figura:

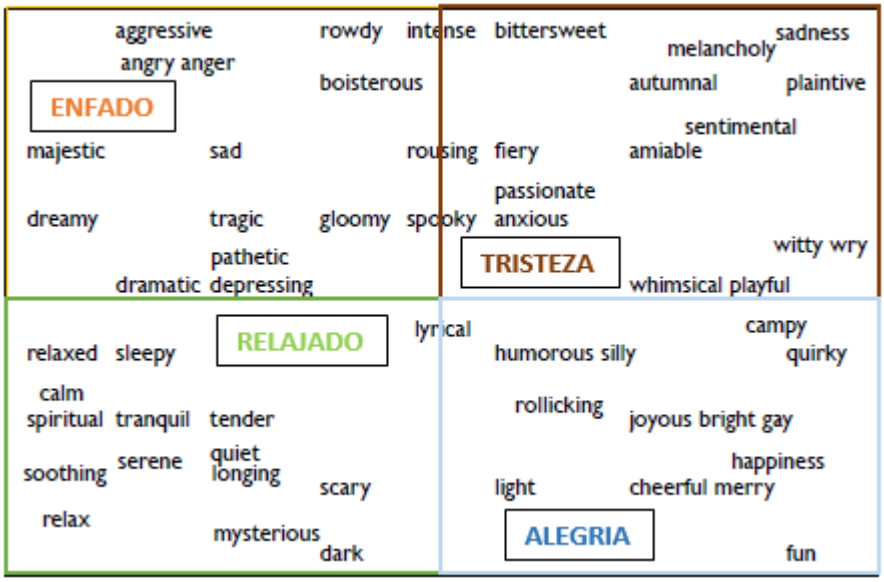

Figura 3-2. Sobre un mapa autoorganizado de las etiquetas de estado de ánimo en el espacio semántico se hacen cuatro grupos asociados a cada estado de ánimo [30].

En la figura se pueden apreciar cuatro grupos más o menos claros, relacionados con las cuatro emociones básicas. Es similar a la representación de Russell.

Para completar la búsqueda de un modelo válido con cuatro emociones básicas, desde la universidad se buscó una representación jerárquica, también relacionada con la valencia y la excitación de las emociones.

Experimentaron con clustering jerárquico aglomerativo común con enlace completo para lograr visualizar las emociones en un diagrama de árbol.

[En el clustering jerárquico los puntos de datos no se asignan a un clúster en un solo paso. Por el contrario, los datos se dividen agregando o dividiendo conjuntos de muestras. En el método aglomerativo, más comúnmente utilizado, las muestras se agrupan por series de fusiones, obteniendo una representación jerárquica en forma de árbol, también llamada dendrograma. El enlace simple, o vecinos más cercanos, define la distancia entre clústeres como la distancia entre el par de muestras más cercano. La técnica de vinculación completa, también llamada vecino más lejano, es lo contrario de la vinculación simple. Las distancias de los clústeres se definen como la distancia entre el par de objetos más distante de cada grupo.]

La solución del proceso de clustering nos lleva hasta el dendrograma mostrado en la Figura:

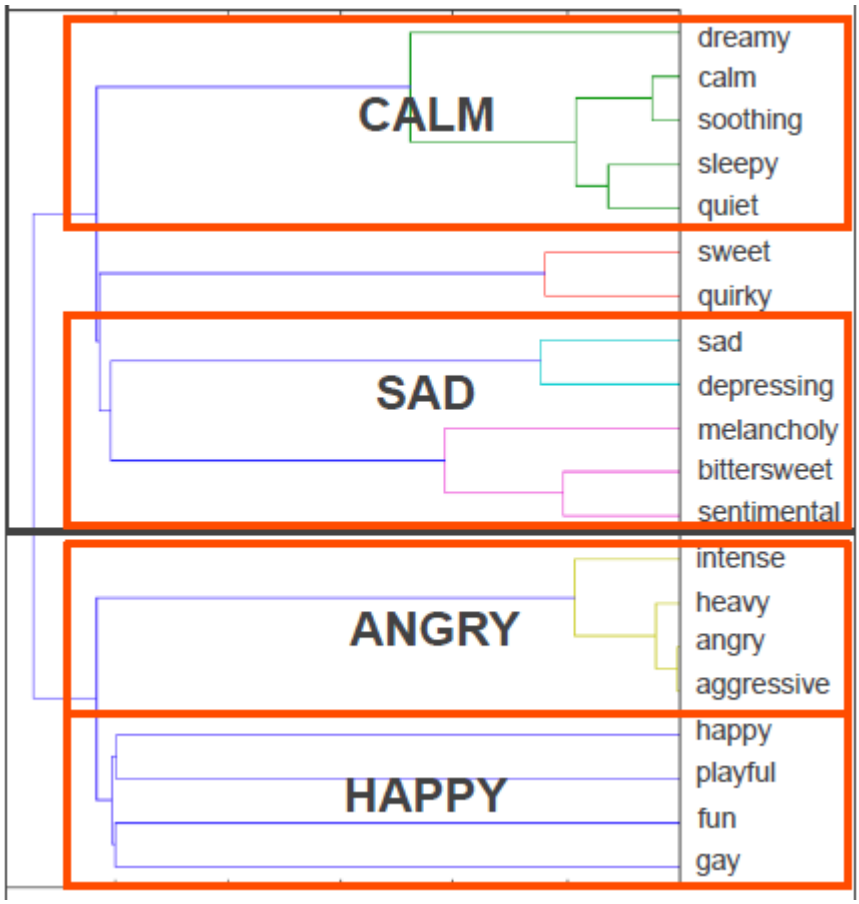

Figura 3-3. Dendrograma de las 20 etiquetas de estado de ánimo más utilizadas, destacando la ramificación correspondiente a la excitación y valencia, mostrando racimos muy relacionados con las emociones básicas [30].

La principal separación de las ramas es gracias a la excitación, separando calma y tristeza, de enfado y felicidad.

Tras varias revisiones, puede darse como aceptado el uso de solo cuatro emociones, las más usadas en una red social.

4 MATERIALES Y MÉTODOS

Solo puedes analizar los datos que tienes. Sé estratégico sobre qué reunir y cómo almacenarlo.

- Marie Curie -

En este capítulo se explicará cada uno de los pasos hasta conseguir la identificación de emociones. Se tomará como referencia el orden marcado para la construcción de un modelo expuesto en el apartado 2.4 de este trabajo.

Se mostrará el dataset utilizado, los cambios realizados sobre los datos para el correcto funcionamiento de los modelos y todos los modelos tratados en el trabajo, explicando cada uno de ellos, enmarcados todos ellos dentro de las redes neuronales.

Los resultados obtenidos y las conclusiones a las que se puedan llegar, aparecerán en capítulos posteriores, cuyo contenido estará íntimamente relacionado e influenciado por este.

Los pasos que aparecerán a continuación son los elegidos para este caso concreto, pero no son exactamente cada uno de los indicados en el apartado 2.4. Es una individualización del proceso de creado de un modelo de Machine Learning, se mantendrá el esquema con matices.

El lenguaje de programación empleado para los códigos es Python, en la versión 3.8.3. El entorno de desarrollo es Jupyter 6.0.3 a través de Anaconda Navigator 1.9.12.

4.1 Dataset

El objetivo es lograr que un modelo de Machine Learning, en concreto un modelo de red neuronal, identifique emociones en la música. El modelo queda bastante definido ya al inicio del proceso y la base de datos se obtiene de forma personal seleccionando canciones a las que después se le reducirá la duración.

El procedimiento de obtención de canciones empieza con una clasificación de las emociones en estilos de música. De una forma muy general y personal, se asocia a la alegría los estilos Disco, Pop Latino o HipHop; al enfado los estilos Hard Rock o Heavy Metal; a la relajación el Blues o el Jazz, y a la tristeza canciones Pop con un ritmo más lento. La línea más fina entre canciones de dos emociones serían las asociadas a la tristeza y a la relajación. A la hora de la selección ambas tienen un ritmo parecido y la diferencia parece residir en que las de tristeza suelen tener letra y el tema de la canción es melancólico.

Para reflejar esta relación y ejemplificar un poco mejor la selección del tipo de canción que aparecerá de forma más marcada en cada grupo, se van a citar dos canciones de cada uno:

- Alegría: Waka Waka (Shakira) y Hey Ya! (OutKast)
- Tristeza: Sober (Demi Lobato) y Aunque tú no lo sepas (Enrique Urquijo)
- Enfado: Murder On (Metallica) y Flick of the sitch (AC/DC)
- Relajado: Do I love you because you're beautiful? (John Coltrane) y That old feeling (Chet Baker)

Una vez elegidos los títulos se buscan en la página de Youtube [32], se copia la url del video en una página que transforma vídeo en mp3 [33] y se obtienen las canciones en mp3. La base de datos final estará formada por 4 grupos, en relación a las 4 emociones elegidas y explicadas en el apartado 3.4, con 120 canciones cada grupo.

El tiempo de duración de una canción común es mucho para procesarlo, por ello se seleccionaron sólo 30 segundos de cada una de las canciones, intentando escoger la parte que mejor refleje la emoción. Como resultado hay cuatro carpetas con 120 canciones de cada una de las emociones de duración 30 segundos.

El siguiente paso es extraer las características de los audios. Esta extracción se ha hecho con la librería "essentia" [34] de la Universidad Pompeu Fabra. La salida final son cuatro archivos en formato json, recopilando las características de todas las canciones.

El json es poco legible, se hace uso del código *23_Ordenar_salida_essentia* para ordenarlo, manteniéndolo en el mismo formato. Dentro del mismo código se efectúa un segundo paso, el de transformar los cuatro archivos json a formato csv para poder trabajar con ellos.

4.2 Selección características

De forma general, si no se conocen las características de los datos con las que se trabajan, la mejor forma para saber si todas son necesarias para el estudio es visualizarlas. En este caso concreto ya existe un estudio previo que extrae para los audios las características de la misma librería essentia y hace un análisis sobre ellas escogiendo las más representativas; por tanto, se seleccionan para el presente trabajo las mismas que se indican en la Tesis inicial [30].

Se ejecuta sobre los cuatro archivos csv obtenidos del paso anterior, que contienen ordenados los atributos de cada emoción, el código *34_Extraer_caracteristicas* y se extraen las siguientes características:

- 'Dissonance'

  Se calcula obteniendo los picos del espectro y midiendo la separación entre ellos. Si los picos son regulares el sonido es consonante y se identifica con relajado, y si son irregulares el sonido es disonante y se asocia al enfado.

  $$Dissonance = \frac{1}{H}\sum_{h} a(h) - SE(h)$$

  dónde H son los armónicos, a(h) la amplitud del armónico y SE(h) la amplitud de la envolvente espectral.

- 'Spectral_Centroid'

  Para separar el enfado, se identifica con valores altos, del no enfado, asociado a valores bajos.

  $$Spectral\ Centroid = \frac{\sum f_i a_i}{\sum a_i}$$

  dónde f es la frecuencia de cada contenedor FFT y a es la amplitud.

- 'Spectral_Complexity'

  Se cuentan los picos espectrales detectados, reflejando así la complejidad de la señal de audio en términos de frecuencia. Útil para diferenciar entre una señal compleja, equivalente al enfado o a la alegría, y señales menos complejas, equivalentes a relajado y triste.

- 'Spectral_Rolloff'

  Las canciones tristes, comparadas con no tristes, tienen un menor valor de Roll-Off.

  $$SpectralRF_t = \max f \left\{ \left| \sum_{n=1}^{f} M_t[n] < TH * \sum_{n=1}^{N} M_t[n] \right| \right\}$$

  dónde $M_t[]$ es la magnitud de la transformada de Fourier y TH es el umbral de energía.

- 'Spectral_Kurtosis'

  Presenta rango más amplio de valores para relajado que para no relajado.

$$Spectral\ Kurtosis = \frac{E(x - \mu)^4}{\sigma^4}$$

  dónde E() es el valor esperado de x, valor normalizado de amplitud, menos μ, media de x. σ es la desviación estándar.

- 'Zero_Crossing_Rate'

  Mide la tasa de cambio de signo de la forma de onda. Útil para distinguir entre enojado, valores altos de ZCR, y relajado, identificado por valores bajos.

$$ZCR = \frac{1}{2}\sum_{n=1}^{N} |sign(x[n]) - sign(x[n-1])|$$

- 'Spectral_Skewness'

  Valores altos indican tristeza y valores bajos lo contrario, canciones no tristes.

$$Spectral\ Skewness = \frac{E(x - \mu)^3}{\sigma^3}$$

  dónde E() es el valor esperado de x, valor normalizado de amplitud, menos μ, media de x. σ es la desviación estándar.

- 'Onset_Rate'

  Definido como el número de inicios en un segundo, siendo el inicio la superación por la energía de un porcentaje específico. Cuando presenta valores altos se asocia a la música alegre y cuando son bajos a la no alegre.

- 'Chords_Scale'

  Número de cambios de acorde por segundo. Los acordes se estiman usando HPCP:

$$HPCP\ (n) = \sum_{i=1}^{nPeaks} w(n, f_i) * a_i^2 \qquad n = 1 \dots N$$

  El vector resultante representa la distribución de energía. Para saber si el modo es mayor o menor, se compara con perfiles de claves de referencia basadas en la teoría musical [35]. El estado de ánimo feliz se relaciona preferentemente con el modo mayor, y el enfado con el modo menor.

Antes de continuar al siguiente paso de transformación de datos, se separan los archivos en el grupo de entrenamiento o Train, y en el grupo de testeo o Test. Primero se separará cada emoción en dos partes, train y test, de forma aleatoria, y después se creará un único csv con los datos de las cuatro emociones para entrenamiento y otro con todos los datos para testeo, dónde también será aleatorio el orden de cada canción.

El proceso de separación del dataset en Train y Test se lleva a cabo con el código *45_Train_Test_Sentimientos*.

4.3 Transformación de los datos

En este trabajo se contaba con una selección de atributos relevantes de las canciones, son los que se han extraído en el paso anterior, pero aun teniendo esta información los datos necesitan de un tratamiento previo al modelado.

Sobre cada uno de los dataset con los que se cuenta en este punto del proceso, el de train y el de test, se van a aplicar los mismos cambios, que son los siguientes, recogidos en el código *56_Transformación*:

4.3.1 Escalado

En este trabajo se emplean redes neuronales y estos modelos necesitan los datos escalados. Esta condición la establecen las funciones de activación, que necesitan la normalización entre (0, 1) y entre (-1, 1) solo para la función de activación tangente hiperbólica.

4.3.2 Etiquetas en binario con variable dummy

Una variable dummy es aquella variable ficticia que toma como valores solo el 0 o el 1. El valor 1 significa afirmación y el valor 0 negación. Esta variable se puede usar para sustituir a una variable categórica, como ocurre en este dataset. La variable categórica que se necesita pasar a 4 variables ficticias es el sentimiento o emoción, que toma como valores alegría, tristeza, enfado y relajado [36].

Al aplicar la función *createDummies* se hace este cambio en el dataset:

Tabla 4-1. Variable categórica antes de transformarla en variable dummy

Mood
alegría
enfado
tristeza
relajado

Tabla 4-2. Variable categórica transformada a dummy

Mood_alegria	Mood_enfado	Mood_relajado	Mood_tristeza
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

4.3.3 Variable Chords_Scale a binario

La variable Chords_Scale es una cadena de texto, tipo de característica no aceptada por la red neuronal, pero no es categórica. Por ello se aplica como transformación la asignación del valor 1 para indicar ‘major’ y el valor 0 para indica ‘minor’.

4.4 Hiperparámetros Red Neuronal

El modelo empleado en este estudio son las redes neuronales y como todo modelo requiere elegir los hiperparámetros que forman parte de la red. La toma de decisión de algunos o la definición de otros aparece en el código 6_*FuncionActivacion_Optimizador*.

- Capa de entrada

  Para su definición necesitamos conocer el número de neuronas de entrada, pero esto viene impuesto por el número de características que lleva cada muestra de entrenamiento, concretamente se necesitan 9.

- Capa de salida

  Quedan definidas sus neuronas según los posibles valores a los que se asignen las salidas. Aquí aparecen 4 emociones, por ello cuatro neuronas de salida.

- Función de activación capa oculta

  Se prueban tres posibilidades que presentan derivada no nula, necesaria esta condición para la retropropagación:

  - Activación Lineal Rectificada: $ReLU\ (x) = \max(0, x)$

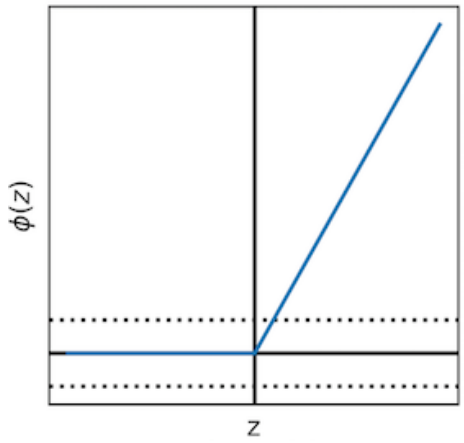

Figura 4-1. Función de activación Lineal Rectificada [37].

  - Logística o Sigmoide: $\sigma\ (x) = \frac{1}{1+ e^{-x}}$

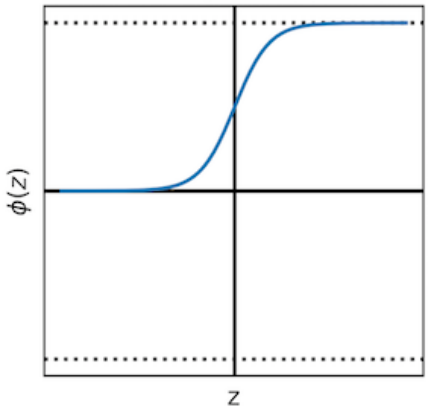

Figura 4-2. Función de activación Logística o Sigmoide [37].

  - Tangente Hiperbólica: $\tanh(x) = \frac{e^{x}- e^{-x}}{e^{x}+ e^{-x}}$

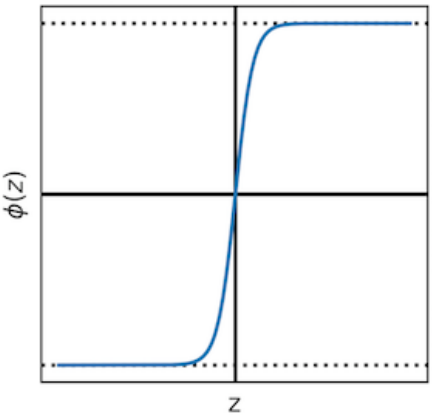

Figura 4-3. Función de activación Tangente Hiperbólica [37].

- Función activación capa salida

  Como las clases son excluyentes y se pretende conseguir una clasificación multiclase, se aplica la función Softmax a la última capa de la red, cuya fórmula es:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$$

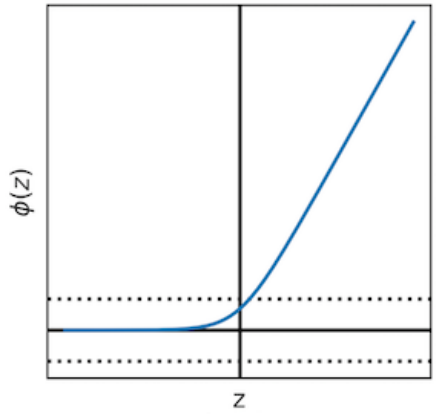

Figura 4-4. Función de activación Softmax [37].

- Optimizador

  Es el encargado de asignar mejores valores a los pesos en cada ejecución para reducir la función de coste, objetivo del entrenamiento de la red. El optimizador calcula para cada peso el gradiente de la función de coste (derivada parcial) y cambia el valor del parámetro hacia la dirección negativa del gradiente [38].

  En el trabajo se ha hecho uso de tres optimizadores diferentes para estudiar su comportamiento con el modelo de red neuronal empleado:

  - SGD o descenso de gradiente estocástico:

    Realiza una iteración con cada muestra, calcula la pérdida y actualiza el peso, siguiendo la fórmula: $pesos = pesos - tasa\ de\ aprendizaje * pérdida$

    La curva de pérdidas puede sufrir fluctuaciones debido al alto número de veces que se ajustan los pesos. Para conseguir una curva más fluida habría que reducir el número de ajustes, pasándole al algoritmo un lote de muestras en lugar de una sola. Se obtiene un promedio de pérdidas del conjunto de muestras y se ajustan los pesos [39].

  - RMSProp.

    Corrige la problemática que pueden presentar otros optimizadores de no llegar nunca al óptimo debido a una desaceleración prematura del gradiente. En lugar de almacenar todos los gradientes que va calculando desde el comienzo de la ejecución, este algoritmo guarda los últimos calculados, mediante el uso de la disminución exponencial en el primer paso [15].

  - Adam.

    Estimación de momento adaptativo y optimizador más empleado en Deep Learning. Funciona con el promedio de gradientes calculados en pasos anteriores que decae exponencialmente, junto con el promedio de gradientes cuadrados [15]. De los gradientes toma el impulso y la varianza, y con la combinación de ambos actualiza los pesos [39].

- Función de coste

  Es la función que va a medir la desviación que existe entre el valor predicho por la red neuronal y el valor real. Cuanto menor sea su valor más eficiente es la red. La minimización de la desviación se logra con el ajuste de los pesos de la red, logrado a través del optimizador [40].

  Para el estudio presente, clasificación multiclase, la función de pérdida elegida es "Cross Entropy" [26]:

$$J(W) = \frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log\left(f_W^k(x^i)\right) + (1 - y_k^i)\log(1 - f_W^k(x^i))\right)$$

- Métrica

  Función usada en la comprobación del rendimiento del modelo con un conjunto de datos no visto hasta el momento, llamado datos de testeo. Estos datos no sirven para entrenar el modelo en relación a la optimización, solo se usan para verificar [39].

  Se elige como medida a la hora de la elección de clase, la precisión "accuracy". La precisión o Accuracy en inglés, es la división entre los elementos correctamente clasificados y el número total de artículos.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  dónde TP es verdadero positivo, TN es verdadero negativo, FP es falso positivo y FN es falso negativo.

  Esta medida es adecuada para este caso de estudio debido a que los datos usados están balanceados y existe el mismo número de muestras de cada emoción.

- Número de capas y número de neuronas por capa.

  Ambos hiperparámetros no se pueden elegir antes de hacer pruebas, sino que es necesario ejecutar varias combinaciones hasta dar con la adecuada para los datos y el objetivo que se tienen. Estos cálculos aparecen en los códigos que se detallan en el próximo capítulo.

4.5 Red Neuronal Multicapa

Las redes neuronales multicapa dan un paso más respecto al perceptrón, en estas aumenta el número de capas ocultas y el número de neuronas por capa. Con estas redes se trabaja en los códigos *8_Red_Neuronal_Multicapas_Sig*, *8_Red_Neuronal_Multicapas_Relu* y *8_Red_Neuronal_Multicapas_Tanh*. El objetivo del código es ver el comportamiento de las redes según el número de capas que presenta con un determinado número de neuronas en cada una de ellas, dependiendo de la función de activación de las capas ocultas. Durante el proceso se evalúan modelos que aumentan y disminuyen sus capas y neuronas.

5 EXPERIMENTOS Y RESULTADOS

Un ser humano debe convertir la información en inteligencia o conocimiento. Hemos tendido a olvidar que ninguna computadora nos hará jamás una nueva pregunta.

- Grace Murray Hopper -

A continuación se van a indicar cada una de las pruebas que se han realizado en la búsqueda de la red óptima. Junto a cada experimento aparecerán los resultados del mismo en una tabla.

Hay un total de 2 pruebas mostradas en los siguientes 3 puntos: función activación y optimizador, relación capas y neuronas, y matriz de confusión para estudiar la clasificación llevada a cabo por la red.

5.1 Experimento 1: función de activación y optimizador

En el primer experimento se van a probar diferentes combinaciones de funciones de activación y optimizadores con la finalidad de ir dando forma a la red neuronal más óptima con la elección de hiperparámetros.

La primera prueba entrena un perceptrón con el optimizador Adam y las funciones de activación Sigmoide, Relu y Tangente Hiperbólica. Se empieza por el optimizador Adam debido a que es el más usado y el que arroja mejores resultados de forma más general. Se entrenan 50, 100 y 150 épocas cada una.

Este primer perceptrón está formado por una capa de entrada con 9 neuronas, son 9 los descriptores que se han obtenido de cada audio de la base de datos, y una capa de salida con 4 neuronas, son 4 las clases a las que se puede asociar cada entrada. Tiene además una capa oculta con 16 neuronas.

De la siguiente ejecución se tomarán las funciones con mejores resultados y el número de épocas, y con estas elecciones se procederá al cambio de optimizador para probar descenso de gradiente estocástico (SGD) y RMSprop.

El proceso aparece en el código *6_FuncionActivacion_Optimizador* y en la siguiente tabla se muestran los resultados de precisión de cada caso:

Tabla 5-1. Precisión del perceptrón con optimizador Adam.

Épocas	**50**	**100**	**150**
Sigmoide	0.7	0.72	0.77
Relu	0.75	0.78	0.77
Tanh	0.72	0.74	0.78

Los mejores resultados respecto a cada función de activación son: Relu con 100 épocas, Sigmoide y Tangente Hiperbólica con 150. Estas 3 combinaciones son las que se probarán a continuación con los otros dos optimizadores y se comparan con Adam. Sigue manteniéndose el mismo modelo de perceptrón que con el optimizador anterior.

Tabla 5-2. Precisión del perceptrón con optimizadores SGD y RMSprop.

Optimizador	**Adam**	**SGD**	**RMSprop**
Sigmoide con 150 épocas	0.77	0.72	0.74
Relu con 100 épocas	0.78	0.76	0.73
Tanh con 150 épocas	0.78	0.69	0.73

Tras los resultados seguimos manteniendo las tres funciones de activación para poder estudiar su respuesta con más capas y neuronas, y el optimizador de Adam, debido a que es el mejor optimizador para las tres funciones. El número de épocas no queda fijado.

5.2 Experimento 2: relación capas y neuronas

En este experimento se sigue haciendo uso de las tres funciones de activación, Sigmoide, Relu y Tanh; se probará variedad de épocas, junto al optimizador Adam, se barajarán el número de capas y el número de neuronas por capa, y se irán aumentando a medida que se vea una mejora en la precisión, o disminuyendo.

Para hacer más cómodo el estudio, se fija una función de activación en cada subapartado, se marca el optimizador Adam, la función de salida es la Softmax, y se van variando las capas, las neuronas y las épocas. Cada prueba con una función de activación va a comenzar con una capa oculta y cuatro posibles valores de neuronas por capa. Estos valores de neuronas por capa se han elegido tomando las primeras cuatro potencias impares de 2, elección aleatoria, descartando el 1. Si los resultados de las pruebas requieren probar otro número de capas o de neuronas, se irán ajustando con cada función.

En cada una de las tablas que se mostrarán a continuación aparecerá el valor x, esta x hará referencia a una estructura de red neuronal no ejecutada con las épocas indicadas debido a que se ha encontrado con esa estructura el valor óptimo. Este valor óptimo aparecerá resaltado en negrita en la tabla.

5.2.1 Función de activación Sigmoide

Para esta primera prueba el código es *8_Red_Neuronal_Multicapas_Sig*. Comienza con una sola capa oculta para cuatro modelos, cada uno de ellos con un número de neuronas por capa diferente. La capa de entrada mantiene sus 9 neuronas y la de salida sus 4 en todos los casos. Se realizaron en total 5 ejecuciones variando el número de épocas, aumentando o disminuyendo según los resultados.

Tabla 5-3. Precisión y pérdida de red neuronal con 1 capa oculta con función Sigmoide.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
25 épocas	x	**loss: 0.5136 - accuracy: 0.7879**	x	x
50 épocas	loss: 0.5392 - accuracy: 0.7475	loss: 0.5356 - accuracy: 0.7677	x	loss: 0.5458 - accuracy: 0.7475
100 épocas	**loss: 0.5287 - accuracy: 0.7576**	loss: 0.5460 - accuracy: 0.7475	loss: 0.6249 - accuracy: 0.7071	**loss: 0.4758 - accuracy: 0.7879**
150 épocas	loss: 0.5447 - accuracy: 0.7273	loss: 0.5533 - accuracy: 0.7273	**loss: 0.4953 - accuracy: 0.7980**	loss: 0.5011 - accuracy: 0.6970
200 épocas	x	x	loss: 0.6011 - accuracy: 0.7374	x

En los resultados de la primera prueba se aprecia que las cuatro opciones barajadas dan buenos resultados, con diferencia en todos los casos en el número de épocas, y destacar que en dos de los casos las pérdidas están por debajo de 0.5.

Para comprobar si un aumento en el número de capas ocultas supone una mejora en la precisión, se repitió el mismo proceso con los mismos modelos a los que se les añadió una capa oculta, teniendo como resultado una estructura con dos capas ocultas.

Tabla 5-4. Precisión y pérdida en red neuronal con 2 capas ocultas con función Sigmoide.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
50 épocas	loss: 0.8909 - accuracy: 0.5051	x	x	loss: 0.5927 - accuracy: 0.6970
100 épocas	**loss: 0.7563 - accuracy: 0.6263**	loss: 0.6147 - accuracy: 0.6869	loss: 0.4965 - accuracy: 0.7475	**loss: 0.4912 - accuracy: 0.7273**
150 épocas	loss: 0.6965 - accuracy: 0.6162	**loss: 0.5275 - accuracy: 0.7273**	**loss: 0.5088 - accuracy: 0.7677**	loss: 0.5305 - accuracy: 0.7273
200 épocas	x	loss: 0.5531 - accuracy: 0.6869	loss: 0.5077 - accuracy: 0.7576	x

Tras la revisión de los resultados de esta segunda prueba se comprobó que en ningún caso se consigue mejorar los valores que se obtuvieron con una sola capa oculta.

Tras confirmar que lo más óptimo es tener un modelo con una capa, se estudió la posibilidad de mejorarlo añadiendo más neuronas. El número de neuronas por capa elegido es la potencia 7 de 2.

Tabla 5-5. Precisión y pérdida en red neuronal con 1 capa oculta con 512 y 1024 neuronas por capa con función Sigmoide.

Número de neuronas en cada capa	**512**	**1024**
50 épocas	loss: 0.5458 - accuracy: 0.7475	loss: 0.5826 - accuracy: 0.7172
100 épocas	**loss: 0.4758 - accuracy: 0.7879**	**loss: 0.6865 - accuracy: 0.7273**
150 épocas	loss: 0.5011 - accuracy: 0.6970	**loss: 0.5809 - accuracy: 0.7273**

El aumento de neuronas tampoco ha dado buenos resultados comparándolos con los que teníamos.

5.2.2 Función de activación Relu

En este segundo subapartado se trabajó con la función de activación Relu en el código 8_*Red_Neuronal_Multicapas_Relu.* La primera estructura de la red vuelve a ser una capa oculta en la cual se irán cambiando el número de neuronas. La capa de entrada mantiene sus 9 neuronas y la de salida sus 4 en todos los casos. El optimizador, cómo ya se avanzó al inicio del capítulo, va a ser siempre Adam.

Tabla 5-6. Precisión y pérdida de red neuronal con 1 capa oculta con función Relu.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
20 épocas	x	x	x	loss: 0.8926 - accuracy: 0.7576
25 épocas	x	x	x	**loss: 0.9392 - accuracy: 0.7879**
50 épocas	loss: 0.5711 - accuracy: 0.7071	loss: 0.5307 - accuracy: 0.7475	loss: 0.5500 - accuracy: 0.7576	loss: 0.6825 - accuracy: 0.7576
100 épocas	loss: 0.5551 - accuracy: 0.7071	loss: 0.4994 - accuracy: 0.7475	**loss: 0.5200 - accuracy: 0.7879**	loss: 0.7181 - accuracy: 0.7576
150 épocas	loss: 0.5615 - accuracy: 0.7172	**loss: 0.5557 - accuracy: 0.7879**	loss: 0.6526 - accuracy: 0.7475	loss: 0.8924 - accuracy: 0.7475
200 épocas	loss: 0.5650 - accuracy: 0.727	loss: 0.6232 - accuracy: 0.7879	x	x
750 épocas	**loss: 0.5685 - accuracy: 0.7677**	x	x	x

800 épocas	loss: 0.6152 - accuracy: 0.7273	x	x	x

Con una sola capa oculta, la red que mejor responde es la que presenta 128 neuronas por capa tras una ejecución de 100 épocas. Muy de cerca, aunque con unas pérdidas un poco más altas, la seguiría la red con 32 neuronas.

Al igual que con la función Sigmoide se comprobó que ocurría con el aumento de capas ocultas, se hará lo mismo usando la función Relu. Se creó un modelo de red con 2 capas ocultas, a las que se les asignó por capa 8, 32, 128 y 512 neuronas. Se ejecutaron dichas redes y los resultados de la precisión se muestran en la siguiente tabla.

Tabla 5-7. Precisión y pérdida en red neuronal con 2 capas ocultas con función Relu.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
50 épocas		loss: 0.5329 - accuracy: 0.7576	loss: 0.5314 - accuracy: 0.7374	loss: 0.5699 - accuracy: 0.7677
100 épocas	loss: 0.4189 - accuracy: 0.8055	**loss: 0.5051 - accuracy: 0.7778**	**loss: 0.6090 - accuracy: 0.7778**	**loss: 0.5332 - accuracy: 0.8283**
150 épocas	**loss: 0.3642 - accuracy: 0.8383**	loss: 0.5247 - accuracy: 0.7677	loss: 0.7005 - accuracy: 0.7778	loss: 0.7173 - accuracy: 0.7677
200 épocas	loss: 0.4296 - accuracy: 0.8095	x	x	x

Tras la ejecución con 2 capas ocultas se observa que todas las estructuras mejoran salvo la que tiene 128 neuronas por capa. Como la mayoría presenta una mejora se van a poner a prueba con una capa más.

Tabla 5-4. Precisión y pérdida en red neuronal con 3 capas ocultas con función Relu.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
25 épocas	loss: 0.6643 - accuracy: 0.6667	loss: 0.7762 - accuracy: 0.6869	loss: 0.6568 - accuracy: 0.7475	**loss: 0.6879 - accuracy: 0.8081**
50 épocas	loss: 0.6239 - accuracy: 0.6667	**loss: 0.4948 - accuracy: 0.7778**	**loss: 0.5650 - accuracy: 0.7778**	**loss: 0.6500 - accuracy: 0.7778**
100 épocas	**loss: 0.5617 - accuracy: 0.7172**	loss: 0.5344 - accuracy: 0.7778	loss: 0.8984 - accuracy: 0.7576	loss: 1.2715 - accuracy: 0.7778
150 épocas	loss: 0.5836 - accuracy: 0.7071	loss: 0.5264 - accuracy: 0.7879	x	x

Con esta última prueba en la que la estructura tiene tres capas ocultas se concluye que el aumento de capas no marca una gran mejoría, y como pasó con la función Sigmoide no se mejora la precisión.

El mejor resultado lo ha dado la red neuronal cuya estructura contaba con 2 capas y 8 neuronas, por ello no se va a buscar mejoría en el aumento de neuronas por capa.

5.2.3 Función de activación Tanh

El código de este subapartado está en *8_Red_Neuronal_Multicapas_Tanh.* Al igual que en los subapartados anteriores la estructura de red cuenta con una capa oculta y optimizador de Adam como estructura inicial. Contará con 9 neuronas en capa de entrada y 4 neuronas en capa de salida. Aquí la función de activación de la capa oculta es Tangente Hiperbólica. Se probará con 8, 32, 128 y 512 neuronas en la capa oculta, y variedad de épocas según resultados.

Tabla 5-9. Precisión y pérdida de red neuronal con 1 capa oculta con función Tangente Hiperbólica.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
25 épocas	x	x	x	loss: 0.5531 - accuracy: 0.7273
50 épocas	loss: 0.4939 - accuracy: 0.7576	loss: 0.5172 - accuracy: 0.7576	loss: 0.6151 - accuracy: 0.7071	**loss: 0.4690 - accuracy: 0.7980**
100 épocas	**loss: 0.4968 - accuracy: 0.7677**	**loss: 0.4720 - accuracy: 0.7879**	loss: 0.5333 - accuracy: 0.7273	loss: 0.6456 - accuracy: 0.7273
150 épocas	loss: 0.4997 - accuracy: 0.7677	loss: 0.5341 - accuracy: 0.7677	loss: 0.5812 - accuracy: 0.7677	x
200 épocas	loss: 0.5279 - accuracy: 0.7576	x	**loss: 0.5340 - accuracy: 0.7778**	x
300 épocas	x	x	loss: 0.6989 - accuracy: 0.7879	x
450 épocas	x	x	loss: 0.8363 - accuracy: 0.7778	x

Tras los resultados con una sola capa oculta se prueba el comportamiento con 2, no se aumenta más debido a que con las anteriores funciones los mejores resultados han sido con 1 o 2. Si fuera necesario se aumentará posteriormente. Todos los demás hiperparámetros serán iguales.

Tabla 5-10. Precisión y pérdida de red neuronal con 2 capas ocultas con función Tangente Hiperbólica.

Número de neuronas en cada capa	**8**	**32**	**128**	**512**
25 épocas	x	x	loss: 0.4965 - accuracy: 0.7374	x
50 épocas	loss: 0.5033 - accuracy: 0.7273	loss: 0.4915 - accuracy: 0.7374	**loss: 0.5177 - accuracy: 0.7980**	loss: 0.7232 - accuracy: 0.7273
100 épocas	**loss: 0.4778 - accuracy: 0.7475**	**loss: 0.4691 - accuracy: 0.7778**	loss: 0.4953 - accuracy: 0.7677	**loss: 0.5243 - accuracy: 0.7879**
150 épocas	loss: 0.5053 - accuracy: 0.7374	loss: 0.4517 - accuracy: 0.7475	x	loss: 0.6271 - accuracy: 0.7677

De los resultados se deduce que las estructuras con 8 y 512 neuronas por capa no mejoran con el aumento de capas ocultas, sin embargo, de las otras hay una que se mantiene en número, la de 32 neuronas, y la de 128 mejora. Por tanto, se toman estas dos últimas y se analizan con 3 capas ocultas.

Tabla 5-11. Precisión y pérdida de red neuronal con 3 capas ocultas con función Tangente Hiperbólica.

Número de neuronas en cada capa	**32**	**128**
50 épocas	loss: 0.5891 - accuracy: 0.7374	loss: 0.5978 - accuracy: 0.7475
100 épocas	**loss: 0.5415 - accuracy: 0.7374**	**loss: 0.5693 - accuracy: 0.7576**
150 épocas	**loss: 0.5611 - accuracy: 0.7576**	loss: 0.7340 - accuracy: 0.7576
250 épocas	loss: 0.9643 - accuracy: 0.7677	x

El aumento de capas no ha dado resultados. Se prueba si es posible con un aumento de neuronas por capa obtener mejores resultados que los de hasta ahora. Salvo el número de capas y de neuronas, lo demás es común al resto de estructuras empleadas en el subapartado.

Tabla 5-12. Precisión y pérdida de red neuronal con 1024 neuronas por capa oculta con función Tangente Hiperbólica.

Número de capas con 1024 neuronas	**1**	**2**	**3**
100 épocas	loss: 0.5579 - accuracy: 0.7576	loss: 0.6097 - accuracy: 0.7172	loss: 0.8434 - accuracy: 0.7071
150 épocas	**loss: 0.5092 - accuracy: 0.7778**	loss: 0.7403 - accuracy: 0.7475	loss: 0.8417 - accuracy: 0.7374
250 épocas	loss: 0.5427 - accuracy: 0.7576	loss: 1.1782 - accuracy: 0.7879	loss: 1.1162 - accuracy: 0.7374

El aumento de neuronas por capa tampoco ha dado muy buen resultado, como se venía apreciando con las funciones de activación anteriores.

5.3 Experimento 3: matriz de confusión

De los mejores resultados obtenidos en cada uno de los subapartados anteriores se va a generar una matriz de confusión para observar lo bien o lo mal que clasifica la red cada una de las emociones y localizar el lugar de una posible mejora.

5.3.1 Matriz de confusión con Sigmoide

Con la función de activación Sigmoide en la capa oculta se obtuvieron los mejores resultados con dos estructuras de redes, ambas con 1 sola capa oculta. La diferencia entre ellas estaba en el número de neuronas de esa capa, 128 y 512, y número de épocas. 150 y 100.

Generamos la matriz para el primer caso:

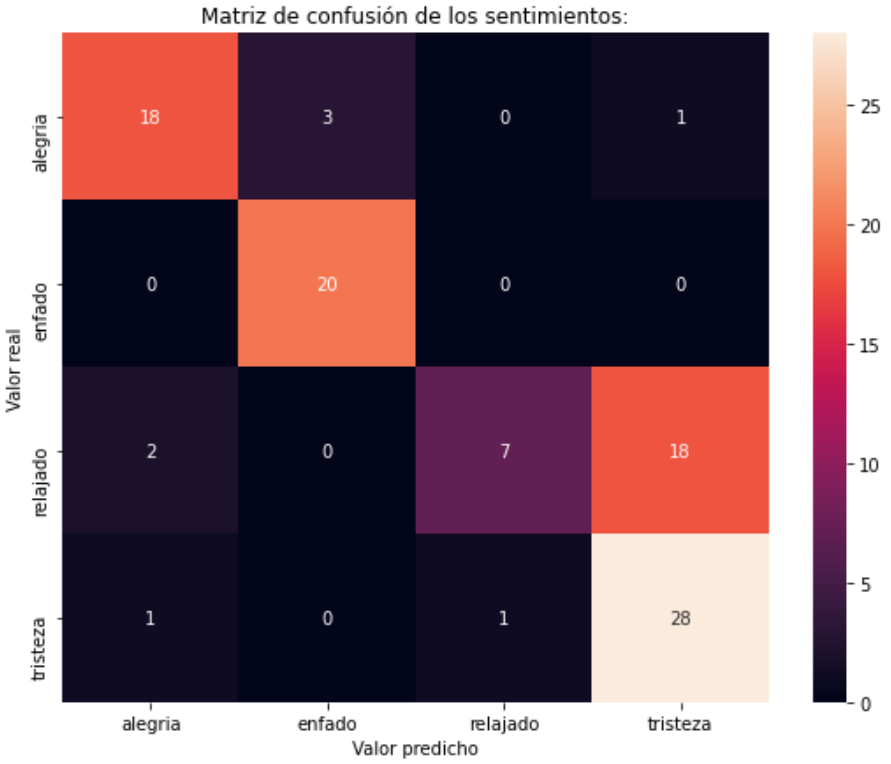

Figura 5-1. Matriz confusión para Sigmoide con 128 neuronas en 1 capa.

Y para el segundo:

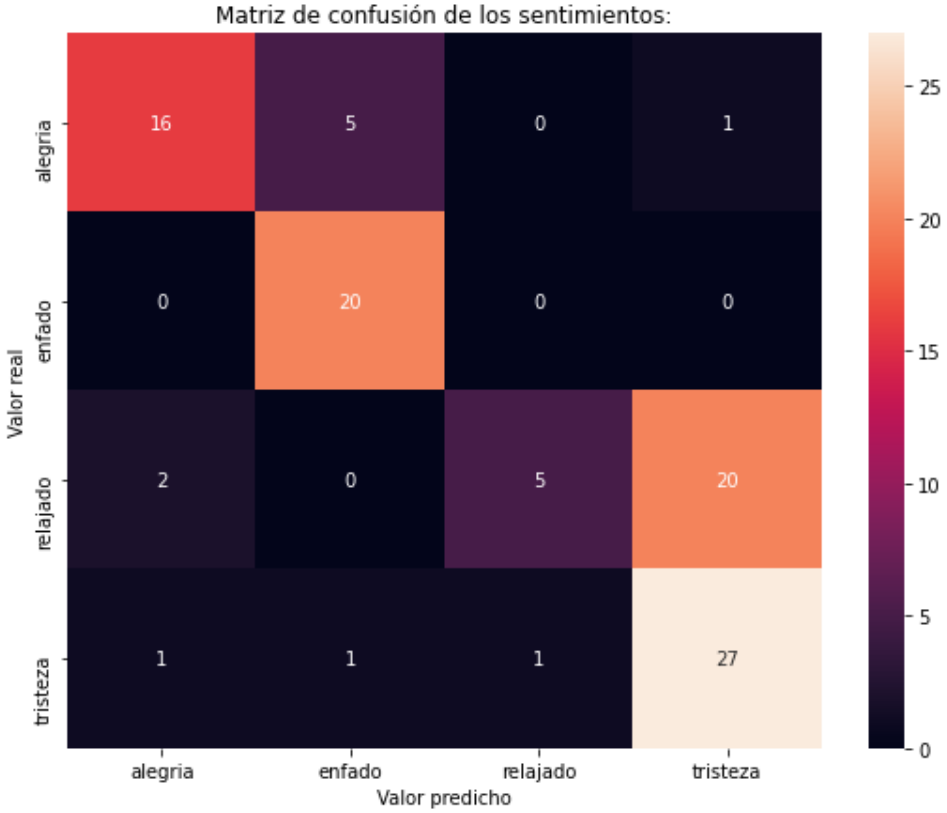

Figura 5-2. Matriz confusión para Sigmoide con 512 neuronas en 1 capa.

Para ambos casos puede deducirse lo mismo, el sentimiento de relajado es el que más le cuesta a la red identificar y lo confunde con la tristeza. El aumento de neuronas por capa tampoco ayuda a elegir mejor.

5.3.2 Matriz de confusión con Relu

En el caso de la función de activación Relu, se toma la mejor estructura para estudiar la matriz de confusión. Esta estructura es la que presenta 2 capas y 8 neuronas en cada una de ellas. A pesar de tener la tasa de error más baja y un acierto de los más levados, se sigue poniendo de manifiesto que la red confunde la emoción relajado con triste:

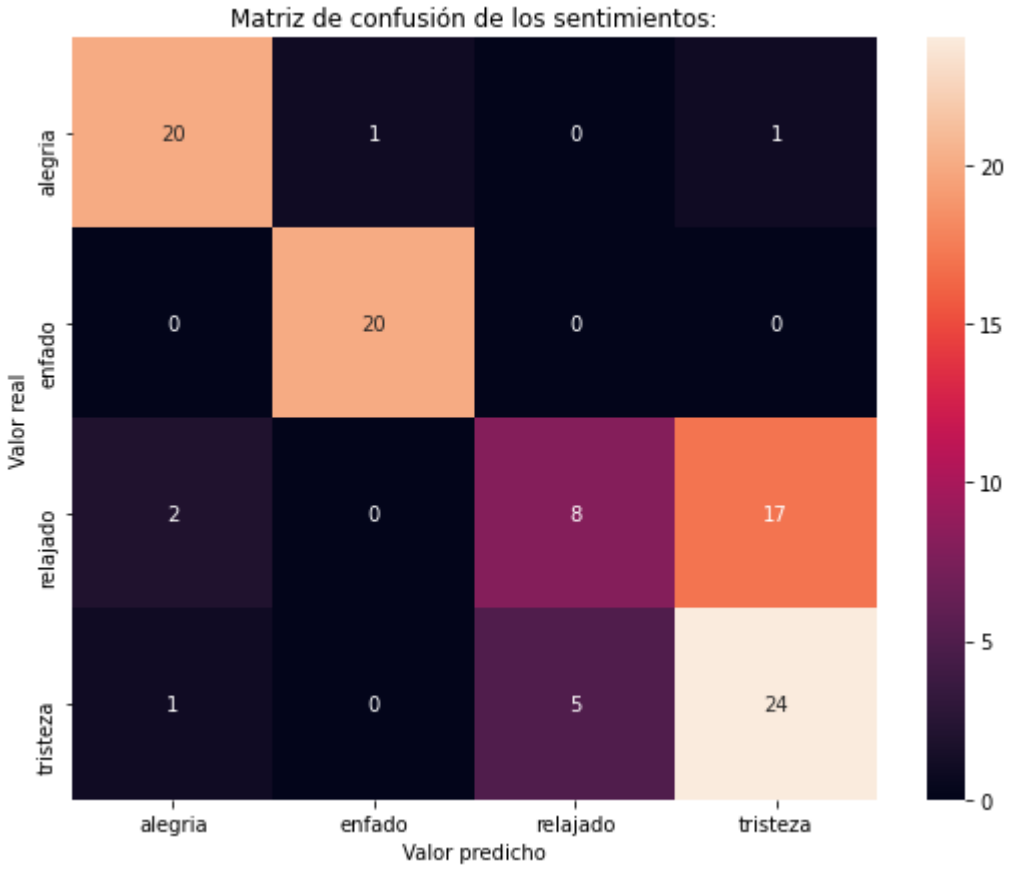

Figura 5-3. Matriz confusión para Relu con 512 neuronas en 2 capas.

5.3.3 Matriz de confusión con Tangente Hiperbólica

Con esta función de activación vuelve a quedar patente que la red mezcla los sentimientos de triste y relajado. Al igual que con la función Sigmoide tiene un alto porcentaje de acierto en todos los sentimientos salvo en relajado.

Se muestra a continuación la matriz de confusión que refleja los falsos positivos que comete el algoritmo:

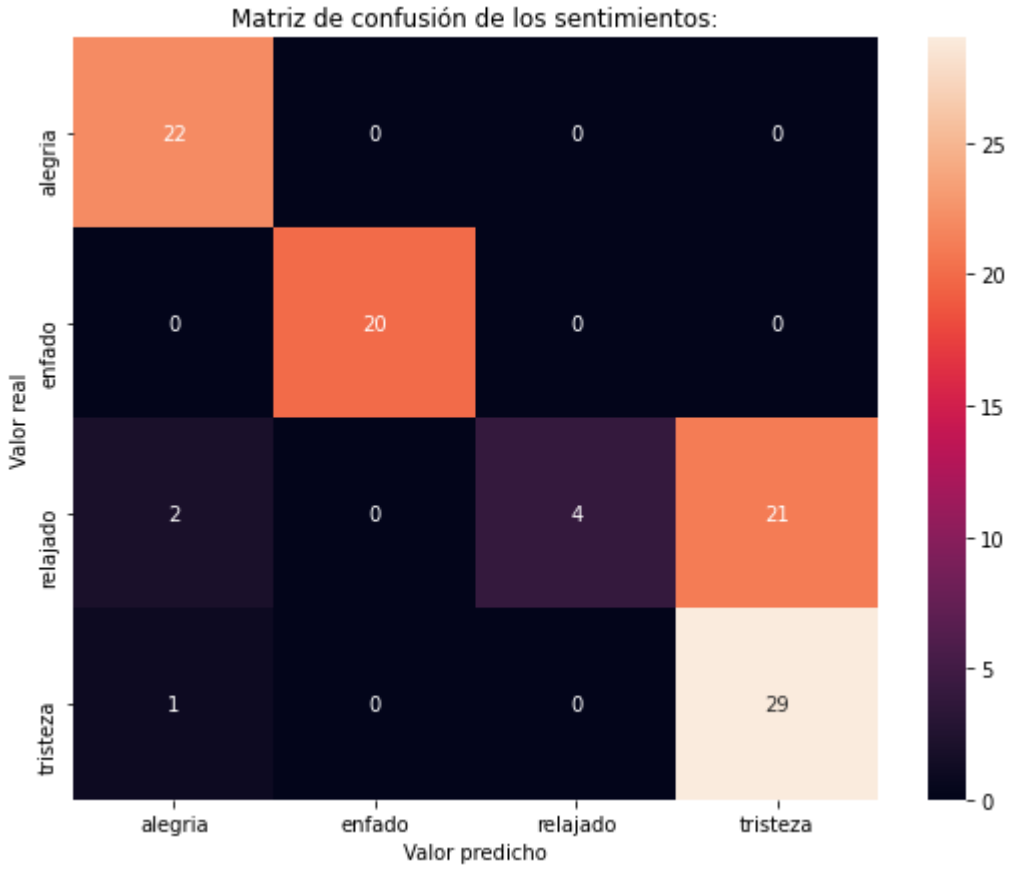

Figura 5-4. Matriz confusión para Tanh con 512 neuronas en 1 capa.

6 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

En este trabajo se han realizado bastantes pruebas para evaluar el comportamiento de una red neuronal con diferentes emociones. A continuación se expondrán las conclusiones alcanzadas y posibles usos futuros teniendo de base este proyecto.

6.1 Conclusiones

Lo primero a destacar es que para la base de datos usada es suficiente con una red sencilla para obtener resultados. El aumento de capas, que aumentaría la complejidad de la red, no hacía que se obtuviera mayor porcentaje de acierto. Y lo mismo ocurría con el aumento de neuronas por capa en las tres estructuras analizadas. Por tanto, con una red sencilla es posible lograr buenos resultados.

Otro aspecto a comentar, como ya parecía intuirse mientras se preparaba la base de datos, es que las emociones de triste y relajado se solapan, más concretamente, el estado de ánimo o sentimiento relajado se asocia prácticamente siempre a la tristeza. No se ha logrado una combinación de capas y neuronas que consiga distinguirlas como mínimo un 50%.

No se han mostrado todas las ejecuciones realizadas para llegar a los datos mostrados en el capítulo anterior, no siempre se conseguía los valores esperados, que son los que se han plasmado.

6.2 Líneas futuras de investigación

Sería interesante conseguir otra base de datos que consiguiera marcar más la diferencia entre las emociones triste y relajado, y observar si es posible lograr diferenciarlas y no confundirlas a la hora de la clasificación. Igualmente se podría probar esto con la misma base de datos pero con otros algoritmos de Machine Learning que sean menos complejos que las redes neuronales, por ejemplo los Árboles de decisión o K-means, para ver si pueden conseguir clasificarlos sin falsos positivos.

Se podría probar a mantener más descriptores de cada audio de la base de datos, al tener más información se podría hacer uso de las redes convolucionales para estudiar su comportamiento a la hora de clasificar emociones y comprobar si existe mejora en la clasificación.

Y lo más interesante es que sirviera de base para futuros proyectos de Inteligencia Artificial, ahora tan presente en todos los ámbitos, y se pudieran lograr avances con esta pareja, emoción y música, incluso en medicina o psicología.

ÍNDICE DE CONCEPTOS

REFERENCIAS

[1] Elaine Thompson, “10 ejemplos de que ya dependes de la IA en tu vida diaria”. OpenMind, 2019. https://www.bbvaopenmind.com/tecnologia/inteligencia-artificial/10-ejemplos-de-que-ya-dependes-de-la-ia-en-tu-vida-diaria/.

[2] Keith Darlingto, “El impacto de la Inteligencia Artificial en la asistencia sanitaria”. OpenMind, 2018. https://www.bbvaopenmind.com/tecnologia/mundo-digital/el-impacto-de-la-inteligencia-artificial-en-la-asistencia-sanitaria/.

[3] Mathworks, “¿Qué es la inteligencia artificial (IA)? Tres cosas que es necesario saber”. Mathworks. https://es.mathworks.com/discovery/artificial-intelligence.html#por-qu%C3%A9-es-importante-la-ia.

[4] GARRIDO, A., 2020. Los avances de la inteligencia artificial . Madrid: Dykinson. ISBN 9788413246604.

[5] Webedia Brand Services, “Los principales hitos en la historia de la inteligencia artificial”. Xakata, 2019. https://ecosistemahuawei.xataka.com/principales-hitos-historia-inteligencia-artificial/

[6] Innovación e inteligencia artificial al servicio del desarrollo rural , 2020. Valencia: Vicerrectorado de Proyección Territorial y Sociedad, Universitàt de València, cop.c2020. ISBN 9788491332657.

[7] Jaime Abad, "Lo más importante en Inteligencia Artificial del 2019". En: Dail Software. Diciembre 2019. Disponible en : https://www.dail.es/progresos-inteligencia-artificial-2019/

[8] Tecnología, "Los principales hitos de la IA en 2020". En: Panda Security Mediacenter. Enero 2021. Disponible en: https://www.pandasecurity.com/es/mediacenter/mobile-news/principales-hitos-ia-2020/

[9] Juan Scaliter, "10 innovaciones tecnológicas que llegarán en 2020". En: National Geographic España. Marzo 2020. Disponible en: https://www.nationalgeographic.com.es/ciencia/10-innovaciones-tecnologicas-que-llegaran-2020_15270/7

[10] Will Douglas Heaven (traducido por Ana Milutinovic), "'Digital twins' con IA para anticipar los fallos de la cadena de suministro". En: MIT Technology Review. Octubre 2021. Disponible en: https://www.technologyreview.es/s/13783/digital-twins-con-ia-para-anticipar-los-fallos-de-la-cadena-de-suministro

[11] Mathworks, “Machine Learning: Tres cosas que es necesario saber”. Mathworks. https://es.mathworks.com/discovery/machine-learning.html#-por-qu%C3%A9-es-importante.

[12] Redacción APD, “¿Qué es el Machine Learning y cómo funciona?”. APD, 2019.
https://www.apd.es/que-es-machine-learning/.

[13] Mathworks, “Deep Learning: Tres cosas que es necesario saber”. Mathworks.

https://es.mathworks.com/discovery/deep-learning.html#howitworks.

[14] LIU, Hongyu; LANG, Bo. Machine learning and deep learning methods for intrusion detection systems: A survey. applied sciences, 2019, vol. 9, no 20, p. 4396.

[15] GÉRON, A., 2019. Hands-on machine learning with scikit-learn, keras, and tensorflow. Sebastopol (CA): O'Reilly. ISBN 9781492032649.

[16] Mathworks, "¿Qué es una red neuronal? Tres cosas que es necesario saber". Mathworks. https://es.mathworks.com/discovery/deep-learning.html#howitworks.

[17] A. Requena, R. Quintanilla, J.M. Bolarín, A. Vázquez, A. Bastida, J. Zúñiga y L.M. Tomá, "Nuevas Tecnologías y Contaminación de Atmósferas, para PYMEs", Universidad de Murcia. https://www.um.es/LEQ/Atmosferas/Ch-VI-3/F63s4p3.htm

[18] F. Arribas Jara, "Aprendizaje no supervisado con modelos generativos profundos," Universidad Autónoma de Madrid, 2018.

[19] "Ejemplo Web Scraping en Python: IBEX35® la Bolsa de Madrid" Aprende Machine Learning, 2019. https://www.aprendemachinelearning.com/ejemplo-web-scraping-python-ibex35-bolsa-valores/.

[20] "Comprende Principal Component Analysis" Aprende Machine Learning, 2018. https://www.aprendemachinelearning.com/comprende-principal-component-analysis/.

[21] "Clasificación con datos desbalanceados" Aprende Machine Learning, 2019. https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/.

[22] TRAPERO ESTEPA, M.D., MURILLO FUENTES, J.J. y PAYÁN SOMET, F.J., 2021. Clasificación de ataques a una red de telecomunicación con Deep Learning Trabajo Fin de Máster . Sevilla: El autor.

[23] "Escalado de datos" Interactive Chaos, 2019. https://interactivechaos.com/es/manual/tutorial-de-machine-learning/escalado-de-datos.

[24] Boyan Angelov, "El impacto de la Inteligencia Artificial en la asistencia sanitaria". Medium, 2017. https://towardsdatascience.com/working-with-missing-data-in-machine-learning-9c0a430df4ce.

[25] "7 pasos del Machine Learning para construir tu máquina.," Aprende Machine Learning, 2017. https://www.aprendemachinelearning.com/7 pasos machine learning construir maquina/.

[26] Martínez del Amor, Miguel Ángel [IEEE Universidad de Sevilla]. "[Webinar] Deep Learning". Youtube <https://www.youtube.com/watch?v=yXuTu1GfHos>

[27] Colaboradores de Wikipedia. Validación cruzada [en línea]. Wikipedia, La enciclopedia libre, 2020 [fecha de consulta: 24 de agosto del 2021]. Disponible en <https://es.wikipedia.org/w/index.php?title=Validaci%C3%B3n_cruzada&oldid=124047241>.

[28] Lacárcel Moreno, J. Psicología de la música y emoción musical. En: *Educatio Siglo XXI, 20, 213-226.* Recuperado a partir de https://revistas.um.es/educatio/article/view/138

[29] Alaminos Fernández, Antonio F. En: *Revista de Ciencias Sociales* Vol. 9, n.º 1, 2014; pp. 15-42 ISSN: 1989-1385 DOI: 10.14198/OBETS2014.9.1.01. Recuperado a partir de https://doaj.org/article/fdda8fb7cbc94ef3aa119c10cb90102e

[30] Laurier, Cyril François. Automatic Classification of musical mood by content-based analysis. 2011 Disponible en: https://repositori.upf.edu/handle/10230/13091?locale-attribute=es

[31] Juslin, P. N. & Sloboda, J. A. (2001). Music and Emotion: Theory and Research. Oxford: Oxford University Press.

[32] Youtube, https://www.youtube.com/

[33] Transformador de enlace de Youtube a mp3, https://youtubetomp3music.com/es26/

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

[35] Krumhansl, C. L. (1997). An exploratory study of musical emotions and psychophysiology. Canadian journal of experimental psychology, 51 (4), 336–353.

[36] Tim Bock, "What are Dummy Variables?". DisplayR, 2018. https://www.displayr.com/what-are-dummy-variables/

[37] *Red Neuronal Multicapa en Keras*. Mariano Rivera, ©2018-2020 [consulta: septiembre 2021]. Disponible en: http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/nn_multicapa/nn_multicapa.html

[38] Luis Velasco, "Optimizadores en redes neuronales profundas: un enfoque práctico". Medium, 2020. https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5

[39] MOOLAYIL, J., 2019. Learn Keras for Deep Neural Networks A Fast-Track Approach to Modern Deep Learning with Python . 1st ed. 2019. Berkeley, CA: Apress. ISBN 1-4842-4240-8.

[40] Ignacio G.R. Gavilán, "Catálogo de componentes de redes neuronales (III): funciones de pérdida". Blog Ignacio G.R. Gavilán, 2020. https://ignaciogavilan.com/catalogo-de-componentes-de-redes-neuronales-iii-funciones-de-perdida/

ANEXO A: CÓDIGOS

23_Ordenar_salida_essentia

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

# ### 23_Ordenar_salida_essentia

# El código "23_Ordenar_salida_essentia" es el encargado primero de ordenar
la salida de la libreria essentia. Ordena las características del archivo
json de salida, y lo mantiene en el mismo formato. La segunda parte del
código transforma de formato json a csv los cuatro archivos con los que se
trabaja.

import json
import pandas as pd

# Se accede a la carpeta dónde se encuentran los archivos json extraidos con
essentia. Y se selecciona la nueva carpeta dónde se desean guardar los
archivos en formato csv, que contienen las caracterísiticas ordenadas.

mainpatch = "C:/Users/Ana Parra/Documents/Trabajo/ANACONDA"

archivos = ['alegria','enfado','relajado','tristeza']

filepatch = 'C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/2_Archivos_salida_essentia/'
ordenpatch = 'C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/3_Archivos_ordenados/'

# El primer bucle ordena el archivo json y lo mantiene en el mismo formato en
la nueva carpeta.

for i in archivos:
    with open(filepatch + i + '.json', "r") as in_data:
        data = json.load(in_data)
    with open(ordenpatch + i + 'ord.json', "w") as out_data:
        json.dump(data ,out_data , sort_keys=True, indent=4, separators=(',',
': '))

# El segundo bucle convierte el archivo json a formato csv.

for j in archivos:
    df = pd.read json (ordenpatch + j + 'ord.json')
    df.to_csv (ordenpatch + j + 'ord.csv', index = None)
```

34_Extraer_caracteristicas

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

# ### 34_Extraer_caracteristicas
```

```
# Este codigo implementa el proceso de extración de un conjunto de
características (tomadas del capítulo 4 de la memoria de referencia de este
trabajo), las mismas para todas las canciones y para cada una de las cuatro
emociones. Tendremos por tanto 9 características de cada canción por cada
emoción en cuatro archivos diferentes.

import json
import csv
import pandas as pd

# Se accede a la carpeta dónde se encuentran los cuatro archivos csv con las
caracteristicas de las 120 canciones de cada emoción. Se selecciona la nueva
carpeta dónde se desean guardar los archivos en formato csv, que contienen la
selección de caracterísitcas.

mainpatch = "C:/Users/Ana Parra/Documents/Trabajo/ANACONDA"
sentimientos = ['alegria','enfado','relajado','tristeza']
filepatch = mainpatch + '/3_Archivos_ordenados/'
featurespatch = mainpatch + '/4_Caracteristicas_por_separado/'

# Se crea un array con las caracterísitcas que se van a extraer.

features =
['Dissonance','Spectral_Centroid','Spectral_Complexity','Spectral_Rolloff','S
pectral_Kurtosis','Zero_Crossing_Rate','Spectral_Skewness','Onset_Rate','Chor
ds_Scale']

features_copy = []

# Abrir cada uno de los cuatro ficheros csv que contienen el conjunto
completo de características.
# Se extraen las caracteristicas y almacenamos en una variable auxiliar.
# Abrir un nuevo fichero csv, donde guardar una copia de esas caracteristicas
seleccionadas extraidas.
#
# Al final del bucle, al final de cada línea, asociada a una canción dentro
del array, se le agrega como nuevo atributo, el sentimiento al que van
asociados todos los atributos.

for i in sentimientos:
    with open(filepatch + i + 'ord.json',"r") as data_file:
        data = json.load(data_file)
        with open (featurespatch + i + '_feature.csv',"w") as f:
            writer = csv.writer(f)

            for cancion in data.keys():
                array_aux = []

                caracteristicas_patch = data[cancion]['stats']['lowlevel']
                array_aux.append(caracteristicas_patch['dissonance']['mean'])

array_aux.append(caracteristicas_patch['spectral_centroid']['mean'])

array_aux.append(caracteristicas_patch['spectral_complexity']['mean'])

array_aux.append(caracteristicas_patch['spectral_rolloff']['mean'])

array_aux.append(caracteristicas_patch['spectral_kurtosis']['mean'])

array_aux.append(caracteristicas_patch['zerocrossingrate']['mean'])

array_aux.append(caracteristicas_patch['spectral_skewness']['mean'])
```

```
                caracteristicas_patch1 = data[cancion]['stats']['rhythm']
                array_aux.append(caracteristicas_patch1['onset_rate'])

                caracteristicas_patch2 = data[cancion]['stats']['tonal']
                array_aux.append(caracteristicas_patch2['chords_scale'])

                array_aux.append(i)

                features_copy.append(array_aux)
                writer.writerows(features_copy)
                features_copy = []
```

45_Train_Test_Sentimientos

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

# ### 45_Train_Test_Sentimientos

# Separar en train y test los cuatro dataset de las caracteristicas de forma
aleatoria, obteniendo de nuevo 8 archivos en formato csv.
# Una vez separados, se juntarán los ocho en dos, uno que englobe a todos los
datos de etrenamiento y otro a todos los datos de testeo.

import csv
import json
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Accedemos a la carpeta de los archivos y a la carpeta dónde irán tras la
separación.

mainpatch = "C:/Users/Ana Parra/Documents/Trabajo/ANACONDA"
sentimientos = ['alegria','enfado','relajado','tristeza']
featurespatch = mainpatch + '/4_Caracteristicas_por_separado/'
train_test_patch = mainpatch + '/5_Dataset_train_test_separados/'
features =
['Dissonance','Spectral_Centroid','Spectral_Complexity','Spectral_Rolloff','S
pectral Kurtosis','Zero Crossing Rate','Spectral Skewness','Onset Rate','Chor
ds_Scale', 'Mood']

# #### Separar en train y test.

# Generar una semilla para mantener el resultado aleatorio de la separación.
np.random.seed(2020)

# Bucle para realizar el proceso de separación de una sola vez para las
cuatro emociones.

for i in sentimientos:
    # Se abre el dataset.
    data = pd.read_csv(featurespatch + i + '_feature.csv', header = None,
names = features)
    # Generar una variable aleatoria que siga una distribución normal, y con
ella hacemos la separación de 80% de datos
    # para entrenamiento, y 20% de datos para testeo.
    var = np.random.randn(len(data))
```

```
    # Tomar los valores del vector var (variable aleatoria) menores que 0.8.
    check = (var<0.8)
    # Se genera el grupo de train y el grupo de test.
    training = data[check]
    testing = data[~check]
    # Se pasa a csv.
    training.to_csv(train_test_patch + i + '_train.csv')
    testing.to_csv(train_test_patch + i + '_test.csv')


# #### Concatenar los dataset de cada sentimiento en uno solo, uno para train
y otro para test

# Abrir los cuatro archivos de train, y guardarlos en df1, df2, df3, df4.
df1 = pd.read_csv(train_test_patch + sentimientos[0] + '_train.csv')
df2 = pd.read_csv(train_test_patch + sentimientos[1] + '_train.csv')
df3 = pd.read_csv(train_test_patch + sentimientos[2] + '_train.csv')
df4 = pd.read_csv(train_test_patch + sentimientos[3] + '_train.csv')
# Una vez abiertos, se concatenan de forma ordenada, es decir, df2, detrás de
df1; df3, detrás de df2, y sucesuvamente.
data_training = pd.concat([df1, df2, df3, df4])

# Una vez obtenido el dataframe de train con las caracteristicas de las
cuatro emociones juntas, se puede aleatorizar el orden
# del archivo csv con la función sample
data_training_random = data_training.sample(frac=1)

# Se transforma el archivo a csv.
data_training_random.to_csv(train_test_patch + 'train.csv')

# Reeptir el proceso para los datos de testeo.
df5 = pd.read_csv(train_test_patch + sentimientos[0] + '_test.csv')
df6 = pd.read_csv(train_test_patch + sentimientos[1] + '_test.csv')
df7 = pd.read_csv(train_test_patch + sentimientos[2] + '_test.csv')
df8 = pd.read_csv(train_test_patch + sentimientos[3] + '_test.csv')
data_testing = pd.concat([df5, df6, df7, df8])
data_testing_random = data_testing.sample(frac=1)
data_testing_random.to_csv(train_test_patch + 'test.csv')
```

56_Transformación

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

# ### 56_Transformación

# En el código siguiente se va a llevar a cabo las transformaciones de
escalado y pasar a binario los atributos que son caracteres. Ambas
transformaciones se hacen sobre el dataset de entrenamiento y el dataset de
testeo.

import seaborn as sns
import numpy as np
import tensorflow as tf
import json
import csv
import pandas as pd

from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import to_categorical

# Cargar dataset de train.
mainpatch = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/5_Dataset_train_test_separados/"
data_train = pd.read_csv(mainpatch + 'train.csv')

# Al tener ahora en un mismos dataset todas las emociones, se puede ver cómo
son los datos de nuestro dataset.
# seaborn proporciona algunas funciones para facilitar esta representación

# El primer paso elimina las dos primeras columnas de indices para la
representación
data_train_pairplot = data_train.iloc[:, 2:12]
sns.pairplot(data_train_pairplot,hue='Mood')

# Histograma
data_train_pairplot.hist()

# #### Escalado
prueba = data_train.iloc[:,2:10]

# Importamos e instanciamos la clase con las opciones por defecto, escalado
entre (0, 1); pero también con la opción del
# escalado entre (-1, 1), necesario para la función de activación tanh que se
usará más adelante.

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler_tanh = MinMaxScaler(feature_range = (-1 , 1))

# Se escalan las características predictivas (no es necesario escalar la
variable objetivo o etiqueta, irá en formato binario).
# De las predictivas, la varibale chord scale no se normaliza, también va en
binario.
# Se entrena el escalador y se transforman las primeras 10 columnas, no
entran en la cuenta as primeras columnas de índices:

transformed_data_train = scaler.fit_transform(data_train.iloc[:,2:10])
transformed_data_train_tanh =
scaler_tanh.fit_transform(data_train.iloc[:,2:10])

# Considerando que el resultado devuelto por el escalador es un array
bidimiensional,
# a continuación podemos reconstruir el dataframe, añadiendo al array en
cuestión la columna "species" con la variable objetivo
# y dando nombre a todas las columnas:

transformed_data_train = pd.DataFrame(transformed_data_train)
transformed_data_train.columns = prueba.columns

transformed_data_train_tanh = pd.DataFrame(transformed_data_train_tanh)
transformed_data_train_tanh.columns = prueba.columns

# Agregar las dos columnas finales del dataframe original.

columnas_finales = data_train.iloc[:,10:]

transformed_data_train = pd.concat([transformed_data_train,columnas_finales],
axis=1)
```

```
transformed_data_train_tanh =
pd.concat([transformed_data_train_tanh,columnas_finales], axis=1)

# #### Etiquetas en binario (variable dummy)

# Keras necesita que todas las etiquetas sean codificadas de forma numérica,
lo que significa, por ejemplo, que hemos de añadir una columna numérica a
cada muestra para representar el estado de ánimo al que pertenece la muestra
de forma binaria (es lo que se llama one-hot-encoding)

def createDummies (df, var_name):
        dummy = pd.get_dummies(df[var_name], prefix = var_name)
        df = df.drop(var_name, axis=1)
        df = pd.concat([df, dummy], axis=1)
        return df

# Con este paso se transforma la columna Mood a binario, para no tener
etiquetas con caracateres

data_train_transformed_dummies = createDummies(transformed_data_train,
"Mood")

data_train_transformed_dummies_tanh =
createDummies(transformed_data_train_tanh, "Mood")

# ### Transformar la variable Chords_Scale a binario

# En lugar de emplear una variable dummies también para esta caracteristica
creando dos nuevas columnas, se puede aplicar una condición que convierta el
valor 'mayor' en 1 y el valor 'minor' en 0, y se matiene una sola columna.

for i in data_train_transformed_dummies.index:
    if data_train_transformed_dummies.loc[i,"Chords_Scale"] == 'major':
        data_train_transformed_dummies.loc[i,"Chords_Scale"] = 1
    else:
        data_train_transformed_dummies.loc[i,"Chords_Scale"] = 0

for i in data_train_transformed_dummies_tanh.index:
    if data_train_transformed_dummies_tanh.loc[i,"Chords_Scale"] == 'major':
        data_train_transformed_dummies_tanh.loc[i,"Chords_Scale"] = 1
    else:
        data_train_transformed_dummies_tanh.loc[i,"Chords_Scale"] = 0

# #### Pasar a csv

# Tras las modificaciones se pasa a csv para poder trabajar con la red
neuronal.
# Vamos a transformarlo, pero para ello no vamos a sobreescribir los que
existen, sino que crearemos nuevos.

mainpatch_red = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/6_Datos_preparados/"

data_train_transformed_dummies.to_csv(mainpatch_red +
'train_transformado.csv')

data_train_transformed_dummies_tanh.to_csv(mainpatch_red +
'train_transformado_tanh.csv')

# Repetimos los cambios del dataset Train al dataset Test para probar la red
neuronal.
```

```
# Leemos el csv
data_test = pd.read_csv(mainpatch + 'test.csv')

# Escalamos los valores.
scaler = MinMaxScaler()
transformed_data_test = scaler.fit_transform(data_test.iloc[:,2:10])
transformed_data_test = pd.DataFrame(transformed_data_test)
prueba = data_test.iloc[:,2:10]
transformed_data_test.columns = prueba.columns
columnas_finales = data_test.iloc[:,10:]
transformed_data_test = pd.concat([transformed_data_test,columnas_finales],
axis=1)

# Creamos la variable dummies para pasar las etiquetas a binario.
data_test_transformed_dummies = createDummies(transformed_data_test, "Mood")

# Cambiamos el valor 'major' por 1 y 'minor' por 0.
for i in data_test_transformed_dummies.index:
    if data_test_transformed_dummies.loc[i,"Chords_Scale"] == 'major':
        data_test_transformed_dummies.loc[i,"Chords_Scale"] = 1
    else:
        data_test_transformed_dummies.loc[i,"Chords_Scale"] = 0

# Tras todas estas modificaciones se pasa a csv.
data_test_transformed_dummies.to_csv(mainpatch_red + 'test_transformado.csv')

# Repetimos los cambios del dataset Train al dataset Test para probar la red
neuronal con función de activación Tanh.

# Escalamos los valores.
scaler_tanh = MinMaxScaler(feature_range = (-1 , 1))
transformed_data_test_tanh =
scaler_tanh.fit_transform(data_test.iloc[:,2:10])
transformed_data_test_tanh = pd.DataFrame(transformed_data_test_tanh)
prueba = data_test.iloc[:,2:10]
transformed_data_test_tanh.columns = prueba.columns
columnas_finales = data_test.iloc[:,10:]
transformed_data_test_tanh =
pd.concat([transformed_data_test_tanh,columnas_finales], axis=1)

# Creamos la variable dummies para pasar las etiquetas a binario.
data_test_transformed_dummies_tanh =
createDummies(transformed_data_test_tanh, "Mood")

# Cambiamos el valor 'major' por 1 y 'minor' por 0.
for i in data_test_transformed_dummies_tanh.index:
    if data_test_transformed_dummies_tanh.loc[i,"Chords_Scale"] == 'major':
        data_test_transformed_dummies_tanh.loc[i,"Chords_Scale"] = 1
    else:
        data_test_transformed_dummies_tanh.loc[i,"Chords_Scale"] = 0

# Tras todas estas modificaciones se pasa a csv
data_test_transformed_dummies_tanh.to_csv(mainpatch_red +
'test_transformado_tanh.csv')
```

6_FuncionActivacion_Optimizador

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

```
# ## 6_FuncionActivacion_Optimizador

# En el siguiente código se va a hacer el estudio de la función de activación
más adecuada para el dataset del que se dispone.

import seaborn as sns
import numpy as np
import tensorflow as tf
import json
import csv
import pandas as pd

from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import to_categorical

# Se toman los dataset transformados y se prueban las distintas posibilidades
de redes.

# ### Obtención de los datos de entrenamiento y testeo

mainpatch_red = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/6_Datos_preparados/"
data_train_red = pd.read_csv(mainpatch_red + 'train_transformado.csv')
data_test_red = pd.read_csv(mainpatch_red + 'test_transformado.csv')
data_train_red_tanh = pd.read_csv(mainpatch_red +
'train_transformado_tanh.csv')
data_test_red_tanh = pd.read_csv(mainpatch_red +
'test_transformado_tanh.csv')

# Separación del dataset train en atributos y etiquetas.
X_train = data_train_red.values[:,1:10]
Y_train = data_train_red.values[:,10:]

# Separación del dataset test en atributos y etiquetas.
X_test = data_test_red.values[:,1:10]
Y_test = data_test_red.values[:,10:]

# Separación del dataset train en atributos y etiquetas. Este dataset lleva
escalado entre (-1, 1).
X_train_tanh = data_train_red_tanh.values[:,1:10]
Y_train_tanh = data_train_red_tanh.values[:,10:]

# Separación del dataset test en atributos y etiquetas. Este dataset lleva
escalado entre (-1, 1).
X_test_tanh = data_test_red_tanh.values[:,1:10]
Y_test_tanh = data_test_red_tanh.values[:,10:]


# ### Construyendo la red neuronal - Sigmoide

# La diferencia más significativa entre redes es cómo especificar la
estructura del modelo (la red neuronal) que queremos usar para el
entrenamiento. En Keras los modelos son más flexibles y hay que indicar el
número de parámetros para determinarlo: número de capas, tamaño de las capas,
tipo de conexiones entre capas, etc.
#
```

```
# Esta primera red es muy simple. Dos de las decisiones vienen impuesta por
los datos: hay 9 características y 4 clases, así que la capa de entrada debe
tener 9 unidades, y la capa de salida debe tener 4 unidades.
#
# Se define el modelo de la forma más sencilla que proporciona Keras, como
una pila de capas (en Keras se dice que es un modelo secuencial).

model_sig = Sequential()

# Las próximas líneas definen el tamaño de la capa de entrada
(input_shape=(9,)), y función de activación de la capa oculta.
model_sig.add(Dense(16, input_shape=(9,)))
model_sig.add(Activation('sigmoid'))

# Tamaño de la capa de salida y la función de activación que usarán sus
unidades.
model_sig.add(Dense(4))
model_sig.add(Activation('softmax'))

# Finalmente, basta especificar la estrategia de optimización de los pesos de
las conexiones de las neuronas
# y la función de pérdida que se intentará minimizar.
# Además, se indica qué métricas se tendrá que considerar para medir el
rendimiento.
model_sig.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=["accuracy"])

# ### Construyendo la red neuronal - Relu

model_relu = Sequential()
model_relu.add(Dense(16, input_shape=(9,)))
model_relu.add(Activation('relu'))
model_relu.add(Dense(4))
model_relu.add(Activation('softmax'))
model_relu.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=["accuracy"])

# ### Construyendo la red neuronal - Tangente Hiperbolica

# La función de activación lleva un escalado de los datos de entrada entre -1
y 1,es el rango en el que aparece la salida.
model_tanh = Sequential()
model_tanh.add(Dense(16, input_shape=(9,)))
model_tanh.add(Activation('tanh'))
model_tanh.add(Dense(4))
model_tanh.add(Activation('softmax'))
model_tanh.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=["accuracy"])


# ### Entrenamiento

# En el paso de construir una red ya quedaba la estructura definida y
compilada.
#
# Entrenar una red neuronal a menudo está relacionado con el concepto de
minibatching, que significa enseñar a la red un subconjunto del dataset
completo, ajustar los pesos respecto a ese subconjunto, y entonces mostrarle
otro subconjunto, repitiendo el proceso. Cuando la red haya visto todos los
datos una vez se denomina epoch (época). Ajustar la relación de
minibatch/epoch depende en gran medida del problema concreto que estamos
abordando.
```

```
#
# En los entrenamientos que se muestran a continuación el valor de batching
es de 1.
#
# Después de entrenar la red, si se quisiera repetir, habría que compilar de
nuevo el modelo para reiniciar los pesos.

# Entrenamiento el modelo con Sigmoide.
model_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss, accuracy = model_sig.evaluate(X_test.astype("float32"),
Y_test.astype("float32"), verbose=0)

# Entrenamiento el modelo con Tanh.
model_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_tanh, accuracy_tanh = model_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"), verbose=0)

# Entrenamiento el modelo con Relu.
model_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_relu, accuracy_relu = model_relu.evaluate(X_test.astype("float32"),
Y_test.astype("float32"), verbose=0)

# Modelo con Sigmoide y 50 épocas
print("Loss_Sig_50 = {:.2f}".format(loss))
print("Accuracy_Sig_50 = {:.2f}".format(accuracy))
print("")
# Modelo con Tanh y 50 épocas
print("Loss_Tanh_50 = {:.2f}".format(loss_tanh))
print("Accuracy_Tanh_50 = {:.2f}".format(accuracy_tanh))
print("")
# Modelo con Relu y 50 épocas
print("Loss_Relu_50 = {:.2f}".format(loss_relu))
print("Accuracy_Relu_50 = {:.2f}".format(accuracy_relu))
# Modelo con Sigmoide y 100 épocas
print("Loss_Sig_100 = {:.2f}".format(loss))
print("Accuracy_Sig_100 = {:.2f}".format(accuracy))
print("")
# Modelo con Tanh y 100 épocas
print("Loss_Tanh_100 = {:.2f}".format(loss_tanh))
print("Accuracy_Tanh_100 = {:.2f}".format(accuracy_tanh))
print("")
# Modelo con Relu y 100 épocas
print("Loss_Relu_100 = {:.2f}".format(loss_relu))
print("Accuracy_Relu_100 = {:.2f}".format(accuracy_relu))
# Modelo con Sigmoide y 150 épocas
print("Loss_Sig_150 = {:.2f}".format(loss))
print("Accuracy_Sig_150 = {:.2f}".format(accuracy))
print("")
# Modelo con Tanh y 150 épocas
print("Loss_Tanh_150 = {:.2f}".format(loss_tanh))
print("Accuracy_Tanh_150 = {:.2f}".format(accuracy_tanh))
print("")
# Modelo con Relu y 150 épocas
```

```
print("Loss_Relu_150 = {:.2f}".format(loss_relu))
print("Accuracy_Relu_150 = {:.2f}".format(accuracy_relu))


# ### Optimizador

# Se toman ejemplos con las funciones de activación Sigmoide, Relu y Tangente
Hiperbólica para comprobar si un optimizador difirente al de Adam es mejor
que este.
# El númerocde épocas será de 100 y 150 por ser las que mejores resultados
arrojan.

# #### Sigmoide con 150 épocas

# Sigmoide + 150 epocas + sgd

model_sig = Sequential()
model_sig.add(Dense(16, input_shape=(9,)))
model_sig.add(Activation('sigmoid'))
model_sig.add(Dense(4))
model_sig.add(Activation('softmax'))
model_sig.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Sigmoide.
# Para el optimizador se recomienda tomar como inputs bloques de datos.
model_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_sig_sgd, accuracy_sig_sgd = model_sig.evaluate(X_test.astype("float32"),
Y_test.astype("float32"), verbose=0)

print("Loss_Sig_Sgd = {:.2f}".format(loss_sig_sgd))
print("Accuracy_Sig_Sgd = {:.2f}".format(accuracy_sig_sgd))
print("")

# Sigmoide + 150 epocas + rmsprop

model_sig = Sequential()
model_sig.add(Dense(16, input_shape=(9,)))
model_sig.add(Activation('sigmoid'))
model_sig.add(Dense(4))
model_sig.add(Activation('softmax'))
model_sig.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Sigmoide.
model_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_sig_rms, accuracy_sig_rms = model_sig.evaluate(X_test.astype("float32"),
Y_test.astype("float32"), verbose=0)

print("Loss_Sig_Rms = {:.2f}".format(loss_sig_rms))
print("Accuracy_Sig_Rms = {:.2f}".format(accuracy_sig_rms))
print("")

# #### Relu con 100 épocas

# Relu + 100 epocas + sgd
```

```
model_relu = Sequential()
model_relu.add(Dense(16, input_shape=(9,)))
model_relu.add(Activation('relu'))
model_relu.add(Dense(4))
model_relu.add(Activation('softmax'))
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Relu.
# Para el optimizador se recomienda tomar como inputs bloques de datos.
model_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=100, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_relu_sgd, accuracy_relu_sgd =
model_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"),
verbose=0)

print("Loss_Relu_Sgd = {:.2f}".format(loss_relu_sgd))
print("Accuracy_Relu_Sgd = {:.2f}".format(accuracy_relu_sgd))
print("")

# Sigmoide + 100 epocas + rmsprop

model_sig = Sequential()
model_sig.add(Dense(16, input_shape=(9,)))
model_sig.add(Activation('relu'))
model_sig.add(Dense(4))
model_sig.add(Activation('softmax'))
model_sig.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Sigmoide.
model_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=100, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_relu_rms, accuracy_relu_rms =
model_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"),
verbose=0)

print("Loss_Relu_Rms = {:.2f}".format(loss_relu_rms))
print("Accuracy_Relu_Rms = {:.2f}".format(accuracy_relu_rms))
print("")

# #### Tanh con 150 épocas

# Tanh + 150 epocas + sgd

model_tanh = Sequential()
model_tanh.add(Dense(16, input_shape=(9,)))
model_tanh.add(Activation('tanh'))
model_tanh.add(Dense(4))
model_tanh.add(Activation('softmax'))
model_tanh.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Tanh.
# Para el optimizador se recomienda tomar como inputs bloques de datos.
model_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=20, verbose=0)
```

```
# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_tanh_sgd, accuracy_tanh_sgd =
model_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"), verbose=0)

print("Loss_Tanh_Sgd = {:.2f}".format(loss_tanh_sgd))
print("Accuracy_Tanh_Sgd = {:.2f}".format(accuracy_tanh_sgd))
print("")

# Tanhb + 150 epocas + rmsprop

model_tanh = Sequential()
model_tanh.add(Dense(16, input_shape=(9,)))
model_tanh.add(Activation('tanh'))
model_tanh.add(Dense(4))
model_tanh.add(Activation('softmax'))
model_tanh.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=["accuracy"])

# Entrenamiento el modelo con Tanh.
model_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=1, verbose=0)

# Función de pérdidas del modelo y la de precisión para evaluarlo.
loss_tanh_rms, accuracy_tanh_rms =
model_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"), verbose=0)

print("Loss_Tanh_Rms = {:.2f}".format(loss_tanh_rms))
print("Accuracy_Tanh_Rms = {:.2f}".format(accuracy_tanh_rms))
print("")
```

8_Red_Neuronal_Multicapas_Sig

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

# ## 8_Red_Neuronal_Multicapas_Sig

# En este código se va a buscar la combinación idónea de capas ocultas y
neuronas.

# #### Importación de librerías y carga de dataset.

import seaborn as sns
import numpy as np
import tensorflow as tf
import json
import csv
import pandas as pd

from tensorflow import keras
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras import optimizers
from keras.utils import to_categorical
from keras.utils import np_utils

# Se toman los dataset con el procesado de los datos, tanto el de train como
el de test.

mainpatch_red = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/6_Datos_preparados/"
data_train_red = pd.read_csv(mainpatch_red + 'train_transformado.csv')
data_test_red = pd.read_csv(mainpatch_red + 'test_transformado.csv')

# Separación del dataset train en atributos y etiquetas.
X_train = data_train_red.values[:,1:10]
Y_train = data_train_red.values[:,10:]

# Separación del dataset test en atributos y etiquetas.
X_test = data_test_red.values[:,1:10]
Y_test = data_test_red.values[:,10:]


# ### Definición del modelo

from keras import models
from keras import layers
from keras.utils.vis_utils import plot_model

# La capa de entrada con 9 neuronas, para 9 características.
# Las capas ocultas van aumentando al igual que las neuronas por capa.
# La capa de salida tendrá 4, por ser 4 emociones.

# 1 capas ocultas con 8, 32, 128 y 512 neuronas -> sigmoide
model_1_8_sig = models.Sequential()
model_1_8_sig.add(layers.Dense(8, activation='sigmoid', input_shape=(9,)))
model_1_8_sig.add(layers.Dense(4, activation='softmax'))

model_1_32_sig = models.Sequential()
model_1_32_sig.add(layers.Dense(32, activation='sigmoid', input_shape=(9,)))
model_1_32_sig.add(layers.Dense(4, activation='softmax'))

model_1_128_sig = models.Sequential()
model_1_128_sig.add(layers.Dense(128, activation='sigmoid',
input_shape=(9,)))
model_1_128_sig.add(layers.Dense(4, activation='softmax'))

model_1_512_sig = models.Sequential()
model_1_512_sig.add(layers.Dense(512, activation='sigmoid',
input_shape=(9,)))
model_1_512_sig.add(layers.Dense(4, activation='softmax'))

#model_2_5.summary()

model_1_8_sig.compile(optimizer='adam',loss='categorical_crossentropy',metric
s=['accuracy'])
model_1_32_sig.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])
model_1_128_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
model_1_512_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
```

```
# ### Proceso de Entrenamiento

# Preparados los datos y definida la red (estructura y funcionalidad),
podemos hacer uso de la instrucción fit para comenzar el proceso de
entrenamiento sobre los datos que tenemos.

model_1_8_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=1)
model_1_32_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=1)
model_1_128_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=1)
model_1_512_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=1)


# ### Proceso de testeo

# Debemos tener en cuenta que los valores mostrados son el error y métricas
calculados sobre los propios datos de entrenamiento. Sin embargo, como el
objetivo de un modelo de aprendizaje es generalizar bien sobre datos que el
proceso de entrenamiento no ha visto anteriormente, necesitamos el conjunto
de test para evaluar cómo se comporta la red sobre ejemplos que no ha usado
para ajustarse.

test_loss_1_8_sig, test_acc_1_8_sig =
model_1_8_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_32_sig, test_acc_1_32_sig =
model_1_32_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_128_sig, test_acc_1_128_sig =
model_1_128_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_512_sig, test_acc_1_512_sig =
model_1_512_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))


# ### Repeticion del proceso aumentando las capas

# 5 capas:

model_5_8_sig = models.Sequential()
model_5_8_sig.add(layers.Dense(8, activation='sigmoid', input_shape=(9,)))
model_5_8_sig.add(layers.Dense(8, activation='sigmoid'))
model_5_8_sig.add(layers.Dense(8, activation='sigmoid'))
model_5_8_sig.add(layers.Dense(8, activation='sigmoid'))
model_5_8_sig.add(layers.Dense(8, activation='sigmoid'))
model_5_8_sig.add(layers.Dense(4, activation='softmax'))

model_5_8_sig.compile(optimizer='adam',loss='categorical_crossentropy',metric
s=['accuracy'])

model_5_32_sig = models.Sequential()
model_5_32_sig.add(layers.Dense(32, activation='sigmoid', input_shape=(9,)))
model_5_32_sig.add(layers.Dense(32, activation='sigmoid'))
model_5_32_sig.add(layers.Dense(32, activation='sigmoid'))
model_5_32_sig.add(layers.Dense(32, activation='sigmoid'))
model_5_32_sig.add(layers.Dense(32, activation='sigmoid'))
model_5_32_sig.add(layers.Dense(4, activation='softmax'))

model_5_32_sig.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])
```

```
model_5_128_sig = models.Sequential()
model_5_128_sig.add(layers.Dense(128, activation='sigmoid',
input_shape=(9,)))
model_5_128_sig.add(layers.Dense(128, activation='sigmoid'))
model_5_128_sig.add(layers.Dense(128, activation='sigmoid'))
model_5_128_sig.add(layers.Dense(128, activation='sigmoid'))
model_5_128_sig.add(layers.Dense(128, activation='sigmoid'))
model_5_128_sig.add(layers.Dense(4, activation='softmax'))

model_5_128_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_5_512_sig = models.Sequential()
model_5_512_sig.add(layers.Dense(512, activation='sigmoid',
input_shape=(9,)))
model_5_512_sig.add(layers.Dense(512, activation='sigmoid'))
model_5_512_sig.add(layers.Dense(512, activation='sigmoid'))
model_5_512_sig.add(layers.Dense(512, activation='sigmoid'))
model_5_512_sig.add(layers.Dense(512, activation='sigmoid'))
model_5_512_sig.add(layers.Dense(4, activation='softmax'))

model_5_512_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_5_8_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=10)
model_5_32_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_5_128_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_5_512_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=10)

loss_5_8_sig, acc_5_8_sig = model_5_8_sig.evaluate(X_test.astype("float32"),
Y_test.astype("float32"))
loss_5_32_sig, acc_5_32_sig =
model_5_32_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_5_128_sig, acc_5_128_sig =
model_5_128_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_5_512_sig, acc_5_512_sig =
model_5_512_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))

# Lo normal es que la red se comporte peor en los datos de test que en los
datos de entrenamiento, ya que el proceso de entrenamiento consiste
precisamente en ajustar los pesos para que el error cometido en estos últimos
se minimice. Esta diferencia de comportamiento entre entrenamiento y test se
denomina overfitting (o sobreajuste).

# ### Aumento o disminición de capas según resultados

# 2 capas:

model_2_8_sig = models.Sequential()
model_2_8_sig.add(layers.Dense(8, activation='sigmoid', input_shape=(9,)))
model_2_8_sig.add(layers.Dense(8, activation='sigmoid'))
model_2_8_sig.add(layers.Dense(4, activation='softmax'))

model_2_8_sig.compile(optimizer='adam',loss='categorical_crossentropy',metric
s=['accuracy'])

model_2_32_sig = models.Sequential()
```

```
model_2_32_sig.add(layers.Dense(32, activation='sigmoid', input_shape=(9,)))
model_2_32_sig.add(layers.Dense(32, activation='sigmoid'))
model_2_32_sig.add(layers.Dense(4, activation='softmax'))

model_2_32_sig.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])

model_2_128_sig = models.Sequential()
model_2_128_sig.add(layers.Dense(128, activation='sigmoid',
input_shape=(9,)))
model_2_128_sig.add(layers.Dense(128, activation='sigmoid'))
model_2_128_sig.add(layers.Dense(4, activation='softmax'))

model_2_128_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_2_512_sig = models.Sequential()
model_2_512_sig.add(layers.Dense(512, activation='sigmoid',
input_shape=(9,)))
model_2_512_sig.add(layers.Dense(512, activation='sigmoid'))
model_2_512_sig.add(layers.Dense(4, activation='softmax'))

model_2_512_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_2_8_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=10)
model_2_32_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_2_128_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_2_512_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=50, batch_size=10)

loss_2_8_sig, acc_2_8_sig = model_2_8_sig.evaluate(X_test.astype("float32"),
Y_test.astype("float32"))
loss_2_32_sig, acc_2_32_sig =
model_2_32_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_2_128_sig, acc_2_128_sig =
model_2_128_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_2_512_sig, acc_2_512_sig =
model_2_512_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))


# ### Elegimos aumentar neuronas para una sola capa

model_1_1024_sig = models.Sequential()
model_1_1024_sig.add(layers.Dense(1024, activation='sigmoid',
input_shape=(9,)))
model_1_1024_sig.add(layers.Dense(4, activation='softmax'))

model_1_1024_sig.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_1_1024_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=100, batch_size=10)
loss_1_1024_sig, acc_1_1024_sig =
model_1_1024_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))


# ### Matriz de confusión 1
```

```
model_1_128_sig = models.Sequential()
model_1_128_sig.add(layers.Dense(128, activation='sigmoid',
input_shape=(9,)))
model_1_128_sig.add(layers.Dense(4, activation='softmax'))

model_1_128_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_1_128_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)

test_loss_1_128_sig, test_acc_1_128_sig =
model_1_128_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))

y_predict=model_1_128_sig.predict(X_test)
cm = confusion_matrix(Y_test.argmax(axis=1), y_predict.argmax(axis=1))
print (cm)

# Realizacion de la matriz de confusion:

cm_df = pd.DataFrame(cm,index = ['alegria','enfado','relajado','tristeza'],
                           columns =
['alegria','enfado','relajado','tristeza'])

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

print ('Accuracy Score :',accuracy_score(Y_test.argmax(axis=1),
y_predict.argmax(axis=1)))
print ('Report : ')
print (classification_report(Y_test.argmax(axis=1), y_predict.argmax(axis=1),

target_names=['alegria','enfado','relajado','tristeza']))
plt.show()


# ### Matriz de confusión 2

model_1_512_sig = models.Sequential()
model_1_512_sig.add(layers.Dense(128, activation='sigmoid',
input_shape=(9,)))
model_1_512_sig.add(layers.Dense(4, activation='softmax'))

model_1_512_sig.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_1_512_sig.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=100, batch_size=10)

test_loss_1_512_sig, test_acc_1_512_sig =
model_1_512_sig.evaluate(X_test.astype("float32"), Y_test.astype("float32"))

y_predict=model_1_512_sig.predict(X_test)
cm = confusion_matrix(Y_test.argmax(axis=1), y_predict.argmax(axis=1))
print (cm)

# Realizacion de la matriz de confusion:

cm_df = pd.DataFrame(cm,index = ['alegria','enfado','relajado','tristeza'],
```

```
                            columns =
['alegria','enfado','relajado','tristeza'])

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

print ('Accuracy Score :',accuracy_score(Y_test.argmax(axis=1),
y_predict.argmax(axis=1)))
print ('Report : ')
print (classification_report(Y_test.argmax(axis=1), y_predict.argmax(axis=1),

target_names=['alegria','enfado','relajado','tristeza']))
plt.show()
```

8_Red_Neuronal_Multicapas_Relu

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

# ## 8_Red_Neuronal_Multicapas_Relu

# En este código se va a buscar la combinación idónea de capas ocultas y
neuronas.

# #### Importación de librerías y carga de dataset.

import seaborn as sns
import numpy as np
import tensorflow as tf

from tensorflow import keras
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from keras import models
from keras import layers
from keras.utils.vis_utils import plot_model

from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras import optimizers
from keras.utils import to_categorical
from keras.utils import np_utils

import json
import csv
import pandas as pd


# Se toman los dataset con el procesado de los datos, tanto el de train como
el de test.
```

```
mainpatch_red = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/6_Datos_preparados/"
data_train_red = pd.read_csv(mainpatch_red + 'train_transformado.csv')
data_test_red = pd.read_csv(mainpatch_red + 'test_transformado.csv')

# Separación del dataset train en atributos y etiquetas.
X_train = data_train_red.values[:,1:10]
Y_train = data_train_red.values[:,10:]

# Separación del dataset test en atributos y etiquetas.
X_test = data_test_red.values[:,1:10]
Y_test = data_test_red.values[:,10:]


# ### Definición del modelo

# La capa de entrada con 9 neuronas, para 9 características.
# Las capas ocultas van aumentando al igual que las neuronas por capa.
# La capa de salida tendrá 4, por ser 4 emociones.

# 1 capas ocultas con 8, 32, 128 y 512 neuronas -> relu
model_1_8_relu = models.Sequential()
model_1_8_relu.add(layers.Dense(8, activation='relu', input_shape=(9,)))
model_1_8_relu.add(layers.Dense(4, activation='softmax'))

model_1_32_relu = models.Sequential()
model_1_32_relu.add(layers.Dense(32, activation='relu', input_shape=(9,)))
model_1_32_relu.add(layers.Dense(4, activation='softmax'))

model_1_128_relu = models.Sequential()
model_1_128_relu.add(layers.Dense(128, activation='relu', input_shape=(9,)))
model_1_128_relu.add(layers.Dense(4, activation='softmax'))

model_1_512_relu = models.Sequential()
model_1_512_relu.add(layers.Dense(512, activation='relu', input_shape=(9,)))
model_1_512_relu.add(layers.Dense(4, activation='softmax'))

model_1_8_relu.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])
model_1_32_relu.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
model_1_128_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])
model_1_512_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])


# ### Proceso de Entrenamiento

# Preparados los datos y definida la red (estructura y funcionalidad),
podemos hacer uso de la instrucción fit para comenzar el proceso de
entrenamiento sobre los datos que tenemos.

model_1_8_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
model_1_32_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
model_1_128_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
model_1_512_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
```

```
# ### Proceso de testeo

# Debemos tener en cuenta que los valores mostrados son el error y métricas
calculados sobre los propios datos de entrenamiento. Sin embargo, como el
objetivo de un modelo de aprendizaje es generalizar bien sobre datos que el
proceso de entrenamiento no ha visto anteriormente, necesitamos el conjunto
de test para evaluar cómo se comporta la red sobre ejemplos que no ha usado
para ajustarse.

test_loss_1_8_relu, test_acc_1_8_relu =
model_1_8_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_32_relu, test_acc_1_32_relu =
model_1_32_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_128_relu, test_acc_1_128_relu =
model_1_128_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
test_loss_1_512_relu, test_acc_1_512_relu =
model_1_512_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))

# ### Aumento o disminición de capas según resultados

# 2 capas:

model_2_8_relu = models.Sequential()
model_2_8_relu.add(layers.Dense(8, activation='relu', input_shape=(9,)))
model_2_8_relu.add(layers.Dense(8, activation='relu'))
model_2_8_relu.add(layers.Dense(4, activation='softmax'))

model_2_8_relu.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])

model_2_32_relu = models.Sequential()
model_2_32_relu.add(layers.Dense(32, activation='relu', input_shape=(9,)))
model_2_32_relu.add(layers.Dense(32, activation='relu'))
model_2_32_relu.add(layers.Dense(4, activation='softmax'))

model_2_32_relu.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_2_128_relu = models.Sequential()
model_2_128_relu.add(layers.Dense(128, activation='relu', input_shape=(9,)))
model_2_128_relu.add(layers.Dense(128, activation='relu'))
model_2_128_relu.add(layers.Dense(4, activation='softmax'))

model_2_128_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_2_512_relu = models.Sequential()
model_2_512_relu.add(layers.Dense(128, activation='relu', input_shape=(9,)))
model_2_512_relu.add(layers.Dense(128, activation='relu'))
model_2_512_relu.add(layers.Dense(4, activation='softmax'))

model_2_512_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_2_8_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
model_2_32_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
model_2_128_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
```

```
model_2_512_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)

loss_2_8_relu, acc_2_8_relu =
model_2_8_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_2_32_relu, acc_2_32_relu =
model_2_32_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_2_128_relu, acc_2_128_relu =
model_2_128_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_2_512_relu, acc_2_512_relu =
model_2_512_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))


# ### Repeticion del proceso aumentando las capas

# 3 capas:

model_3_8_relu = models.Sequential()
model_3_8_relu.add(layers.Dense(8, activation='relu', input_shape=(9,)))
model_3_8_relu.add(layers.Dense(8, activation='relu'))
model_3_8_relu.add(layers.Dense(8, activation='relu'))
model_3_8_relu.add(layers.Dense(4, activation='softmax'))

model_3_8_relu.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])

model_3_32_relu = models.Sequential()
model_3_32_relu.add(layers.Dense(32, activation='relu', input_shape=(9,)))
model_3_32_relu.add(layers.Dense(32, activation='relu'))
model_3_32_relu.add(layers.Dense(32, activation='relu'))
model_3_32_relu.add(layers.Dense(4, activation='softmax'))

model_3_32_relu.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])

model_3_128_relu = models.Sequential()
model_3_128_relu.add(layers.Dense(128, activation='relu', input_shape=(9,)))
model_3_128_relu.add(layers.Dense(128, activation='relu'))
model_3_128_relu.add(layers.Dense(128, activation='relu'))
model_3_128_relu.add(layers.Dense(4, activation='softmax'))

model_3_128_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_3_512_relu = models.Sequential()
model_3_512_relu.add(layers.Dense(512, activation='relu', input_shape=(9,)))
model_3_512_relu.add(layers.Dense(512, activation='relu'))
model_3_512_relu.add(layers.Dense(512, activation='relu'))
model_3_512_relu.add(layers.Dense(4, activation='softmax'))

model_3_512_relu.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_3_8_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_3_32_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_3_128_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
model_3_512_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=200, batch_size=10)
```

```
loss_3_8_relu, acc_3_8_relu =
model_3_8_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_3_32_relu, acc_3_32_relu =
model_3_32_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_3_128_relu, acc_3_128_relu =
model_3_128_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))
loss_3_512_relu, acc_3_512_relu =
model_3_512_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))


# Lo normal es que la red se comporte peor en los datos de test que en los
datos de entrenamiento, ya que el proceso de entrenamiento consiste
precisamente en ajustar los pesos para que el error cometido en estos últimos
se minimice. Esta diferencia de comportamiento entre entrenamiento y test se
denomina overfitting (o sobreajuste).

# ### Matriz de confusión 1

# 2 capas con 8 neuronas
model_2_8_relu = models.Sequential()
model_2_8_relu.add(layers.Dense(8, activation='relu', input_shape=(9,)))
model_2_8_relu.add(layers.Dense(8, activation='relu'))
model_2_8_relu.add(layers.Dense(4, activation='softmax'))

model_2_8_relu.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])

model_2_8_relu.fit(X_train.astype("float32"), Y_train.astype("float32"),
epochs=150, batch_size=10)
loss_2_8_relu, acc_2_8_relu =
model_2_8_relu.evaluate(X_test.astype("float32"), Y_test.astype("float32"))

y_predict = model_2_8_relu.predict(X_test)
cm = confusion_matrix(Y_test.argmax(axis=1), y_predict.argmax(axis=1))
print (cm)

# Realizacion de la matriz de confusion:

cm_df = pd.DataFrame(cm,index = ['alegria','enfado','relajado','tristeza'],
                          columns =
['alegria','enfado','relajado','tristeza'])

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

print ('Accuracy Score :',accuracy_score(Y_test.argmax(axis=1),
y_predict.argmax(axis=1)))
print ('Report : ')
print (classification_report(Y_test.argmax(axis=1), y_predict.argmax(axis=1),

target_names=['alegria','enfado','relajado','tristeza']))
plt.show()
```

8_Red_Neuronal_Multicapas_Tanh

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

```
# ## 8_Red_Neuronal_Multicapas_Tanh

# En este código se va a buscar la combinación idónea de capas ocultas y
neuronas.

# #### Importación de librerías y carga de dataset.

import seaborn as sns
import numpy as np
import tensorflow as tf
import json
import csv
import pandas as pd

from tensorflow import keras
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras import optimizers
from keras.utils import to_categorical
from keras.utils import np_utils

# Se toman los dataset con el procesado de los datos para Tanh, tanto el de
train como el de test.

mainpatch_red = "C:/Users/Ana
Parra/Documents/Trabajo/ANACONDA/6_Datos_preparados/"
data_train_red_tanh = pd.read_csv(mainpatch_red +
'train_transformado_tanh.csv')
data_test_red_tanh = pd.read_csv(mainpatch_red +
'test_transformado_tanh.csv')

# Separación del dataset train en atributos y etiquetas. Este dataset lleva
escalado entre (-1, 1).
X_train_tanh = data_train_red_tanh.values[:,1:10]
Y_train_tanh = data_train_red_tanh.values[:,10:]

# Separación del dataset test en atributos y etiquetas. Este dataset lleva
escalado entre (-1, 1).
X_test_tanh = data_test_red_tanh.values[:,1:10]
Y_test_tanh = data_test_red_tanh.values[:,10:]


# ### Definición del modelo

from keras import models
from keras import layers
from keras.utils.vis_utils import plot_model

# La capa de entrada con 9 neuronas, para 9 características.
# Las capas ocultas van aumentando al igual que las neuronas por capa.
# La capa de salida tendrá 4, por ser 4 emociones.

# 1 capas ocultas con 8, 32, 128 y 512 neuronas -> tanh
```

```
model_1_8_tanh = models.Sequential()
model_1_8_tanh.add(layers.Dense(8, activation='tanh', input_shape=(9,)))
model_1_8_tanh.add(layers.Dense(4, activation='softmax'))

model_1_32_tanh = models.Sequential()
model_1_32_tanh.add(layers.Dense(32, activation='tanh', input_shape=(9,)))
model_1_32_tanh.add(layers.Dense(4, activation='softmax'))

model_1_128_tanh = models.Sequential()
model_1_128_tanh.add(layers.Dense(128, activation='tanh', input_shape=(9,)))
model_1_128_tanh.add(layers.Dense(4, activation='softmax'))

model_1_512_tanh = models.Sequential()
model_1_512_tanh.add(layers.Dense(512, activation='tanh', input_shape=(9,)))
model_1_512_tanh.add(layers.Dense(4, activation='softmax'))

#model_2_5.summary()

model_1_8_tanh.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])
model_1_32_tanh.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
model_1_128_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])
model_1_512_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])


# ### Proceso de Entrenamiento

# Preparados los datos y definida la red (estructura y funcionalidad),
podemos hacer uso de la instrucción fit para comenzar el proceso de
entrenamiento sobre los datos que tenemos.

model_1_8_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_1_32_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_1_128_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_1_512_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=25, batch_size=1)


# ### Proceso de testeo

# Debemos tener en cuenta que los valores mostrados son el error y métricas
calculados sobre los propios datos de entrenamiento. Sin embargo, como el
objetivo de un modelo de aprendizaje es generalizar bien sobre datos que el
proceso de entrenamiento no ha visto anteriormente, necesitamos el conjunto
de test para evaluar cómo se comporta la red sobre ejemplos que no ha usado
para ajustarse.

test_loss_1_8_tanh, test_acc_1_8_tanh =
model_1_8_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_1_32_tanh, test_acc_1_32_tanh =
model_1_32_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_1_128_tanh, test_acc_1_128_tanh =
model_1_128_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
```

```
test_loss_1_512_tanh, test_acc_1_512_tanh =
model_1_512_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))


# ### Repeticion del proceso aumentando las capas

# 2 capas:

model_2_8_tanh = models.Sequential()
model_2_8_tanh.add(layers.Dense(8, activation='tanh', input_shape=(9,)))
model_2_8_tanh.add(layers.Dense(8, activation='tanh'))
model_2_8_tanh.add(layers.Dense(4, activation='softmax'))

model_2_32_tanh = models.Sequential()
model_2_32_tanh.add(layers.Dense(32, activation='tanh', input_shape=(9,)))
model_2_32_tanh.add(layers.Dense(32, activation='tanh'))
model_2_32_tanh.add(layers.Dense(4, activation='softmax'))

model_2_128_tanh = models.Sequential()
model_2_128_tanh.add(layers.Dense(128, activation='tanh', input_shape=(9,)))
model_2_128_tanh.add(layers.Dense(128, activation='tanh'))
model_2_128_tanh.add(layers.Dense(4, activation='softmax'))

model_2_512_tanh = models.Sequential()
model_2_512_tanh.add(layers.Dense(512, activation='tanh', input_shape=(9,)))
model_2_512_tanh.add(layers.Dense(512, activation='tanh'))
model_2_512_tanh.add(layers.Dense(4, activation='softmax'))

model_2_8_tanh.compile(optimizer='adam',loss='categorical_crossentropy',metri
cs=['accuracy'])
model_2_32_tanh.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
model_2_128_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])
model_2_512_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_2_8_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_2_32_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_2_128_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=25, batch_size=1)
model_2_512_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)

test_loss_2_8_tanh, test_acc_2_8_tanh =
model_2_8_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_2_32_tanh, test_acc_2_32_tanh =
model_2_32_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_2_128_tanh, test_acc_2_128_tanh =
model_2_128_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_2_512_tanh, test_acc_2_512_tanh =
model_2_512_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))

# Lo normal es que la red se comporte peor en los datos de test que en los
datos de entrenamiento, ya que el proceso de entrenamiento consiste
```

```
precisamente en ajustar los pesos para que el error cometido en estos últimos
se minimice. Esta diferencia de comportamiento entre entrenamiento y test se
denomina overfitting (o sobreajuste).

# ### Aumento o disminición de capas según resultados

# 3 capas:

model_3_32_tanh = models.Sequential()
model_3_32_tanh.add(layers.Dense(32, activation='tanh', input_shape=(9,)))
model_3_32_tanh.add(layers.Dense(32, activation='tanh'))
model_3_32_tanh.add(layers.Dense(32, activation='tanh'))
model_3_32_tanh.add(layers.Dense(4, activation='softmax'))

model_3_128_tanh = models.Sequential()
model_3_128_tanh.add(layers.Dense(128, activation='tanh', input_shape=(9,)))
model_3_128_tanh.add(layers.Dense(128, activation='tanh'))
model_3_128_tanh.add(layers.Dense(128, activation='tanh'))
model_3_128_tanh.add(layers.Dense(4, activation='softmax'))

model_3_32_tanh.compile(optimizer='adam',loss='categorical_crossentropy',metr
ics=['accuracy'])
model_3_128_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_3_32_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)
model_3_128_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=150, batch_size=10)

test_loss_3_32_tanh, test_acc_3_32_tanh =
model_3_32_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_3_128_tanh, test_acc_3_128_tanh =
model_3_128_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))


# ### Elegimos aumentar neuronas para varias capas

model_1_1024_tanh = models.Sequential()
model_1_1024_tanh.add(layers.Dense(1024, activation='tanh',
input_shape=(9,)))
model_1_1024_tanh.add(layers.Dense(4, activation='softmax'))

model_1_1024_tanh.compile(optimizer='adam',loss='categorical_crossentropy',me
trics=['accuracy'])

model_2_1024_tanh = models.Sequential()
model_2_1024_tanh.add(layers.Dense(1024, activation='tanh',
input_shape=(9,)))
model_2_1024_tanh.add(layers.Dense(1024, activation='tanh'))
model_2_1024_tanh.add(layers.Dense(4, activation='softmax'))

model_2_1024_tanh.compile(optimizer='adam',loss='categorical_crossentropy',me
trics=['accuracy'])

model_3_1024_tanh = models.Sequential()
model_3_1024_tanh.add(layers.Dense(1024, activation='tanh',
input_shape=(9,)))
model_3_1024_tanh.add(layers.Dense(1024, activation='tanh'))
model_3_1024_tanh.add(layers.Dense(1024, activation='tanh'))
```

```
model_3_1024_tanh.add(layers.Dense(4, activation='softmax'))

model_3_1024_tanh.compile(optimizer='adam',loss='categorical_crossentropy',me
trics=['accuracy'])

model_1_1024_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=250, batch_size=10)
model_2_1024_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=250, batch_size=10)
model_3_1024_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=250, batch_size=10)

test_loss_1_1024_tanh, test_acc_1_1024_tanh =
model_1_1024_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_2_1024_tanh, test_acc_2_1024_tanh =
model_2_1024_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))
test_loss_3_1024_tanh, test_acc_3_1024_tanh =
model_3_1024_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))


# ### Matriz de confusión 1

# 1 capa con 512 neuronas
model_1_512_tanh = models.Sequential()
model_1_512_tanh.add(layers.Dense(128, activation='tanh', input_shape=(9,)))
model_1_512_tanh.add(layers.Dense(4, activation='softmax'))

model_1_512_tanh.compile(optimizer='adam',loss='categorical_crossentropy',met
rics=['accuracy'])

model_1_512_tanh.fit(X_train_tanh.astype("float32"),
Y_train_tanh.astype("float32"), epochs=50, batch_size=1)

test_loss_1_512_tanh, test_acc_1_512_tanh =
model_1_512_tanh.evaluate(X_test_tanh.astype("float32"),
Y_test_tanh.astype("float32"))

y_predict = model_1_512_tanh.predict(X_test_tanh)
cm = confusion_matrix(Y_test_tanh.argmax(axis=1), y_predict.argmax(axis=1))
print (cm)

# Realizacion de la matriz de confusion:

cm_df = pd.DataFrame(cm,index = ['alegria','enfado','relajado','tristeza'],
                           columns =
['alegria','enfado','relajado','tristeza'])

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

print ('Accuracy Score :',accuracy_score(Y_test_tanh.argmax(axis=1),
y_predict.argmax(axis=1)))
print ('Report : ')
print (classification_report(Y_test_tanh.argmax(axis=1),
y_predict.argmax(axis=1),
```

```
target_names=['alegria','enfado','relajado','tristeza']))
plt.show()
```