UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

CARACTERIZACIÓN DE MÚSICA SEGÚN EMOCIONES Y COMPLEJIDAD, UTILIZANDO RNN-LSTM Y TEORÍA DE LA INFORMACIÓN, PARA ANALIZAR SUS EFECTOS SOBRE LA EMPATÍA HACIA EL DOLOR.

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

LEONARDO ISMAEL PEÑA PEÑA

PROFESOR GUÍA:
LEONEL EUGENIO MEDINA DAZA

MIEMBROS DE LA COMISIÓN:
ANDRÉS EDUARDO CABA RUTTE
RENÉ IGNACIO FERNANDO SAN MARTÍN ULLOA

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: LEONARDO ISMAEL PEÑA PEÑA
FECHA: 2019
PROF. GUÍA: LEONEL EUGENIO MEDINA DAZA

CARACTERIZACIÓN DE MÚSICA SEGÚN EMOCIONES Y COMPLEJIDAD, UTILIZANDO RNN-LSTM Y TEORÍA DE LA INFORMACIÓN, PARA ANALIZAR SUS EFECTOS SOBRE LA EMPATÍA HACIA EL DOLOR.

La empatía en la humanidad es un elemento fundamental para construir una sociedad justa. A su vez la empatía puede ser modulada por diferentes factores, como la emoción que tiene un individuo. Por ende la música, como detonante de emociones en el humano, es capaz de modular la empatía. Al mismo tiempo, se postula que la complejidad que tiene la música, en conjunto con la capacidad que tiene un individuo para percibir diferentes grados de complejidad de ésta, podría modular también, en el cerebro, la respuesta empática que tienen las personas. Se propone en este trabajo diferentes medidores para evaluar la emoción y la complejidad que tienen ciertas piezas musicales. Esta información se pone a disposición, junto con el diseño de un experimento que las utiliza, a la investigación psicológica acerca de el efecto de la musica en la respuesta empática de las personas. En cuanto a las emociones, se presenta un enfoque que utiliza aprendizaje de máquinas, específicamente RNN-LSTM para la predicción de las emociones que evoca la música en un sujeto mientras la escucha. En dicho trabajo se obtuvo 0.8 en el promedio de los errores de test. Por otro lado, en lo referente a las complejidades, se aplican a diferentes repertorios de música clásica de los siglos XVII y XVIII, diferentes medidas de la información, tales como la entropía de primer orden, la entropía condicional y entropía normalizada, para luego, en base a un análisis cualitativo, evaluar qué medida, aplicada a que aspecto de las partituras de cada repertorio, es el que mejor representa la complejidad en la música, resultando que es la entropía condicional, la cual posiciona a .El clavecín bien temperado"de Bach como el repertorio más complejo y a "Los cuartetos de barberíaçomo el menos complejo. En el experimento propuesto se toman dichas características y se realiza un EEG mientras los sujetos escuchan la música caracterizada y ven imágenes con y sin contenido de dolor, además de que responden cuestionarios relacionados a la empatía y a la música. Con esta información se espera verificar la existencia de algún tipo de correlación entre las características extraídas de la música y la respuesta empática hacia el dolor. En síntesis, este trabajo intenta fundamentalmente aportar herramientas ingenieriles a la investigación acerca de cómo afecta la música en la respuesta empática de las personas.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

En todas las culturas humanas se ha hecho presente la música en sus diversas expresiones. Asombro ha causado encontrar vestigios de instrumentos musicales en diferentes lugares del planeta tierra, así como, observar e interpretar el arte rupestre que, ilustrando rituales o ceremonias que muestran a personas bailando, cantando y percutiendo instrumentos, nos permiten entender que, la música, como manifestación cultural, es un *instrumento* de expresión eminentemente humana, cual ha estado siempre presente en el devenir y el desarrollo de la humanidad.

En el libro *El mundo de la música* [86] se plantea que *"La música tiene su origen"... "en la necesidad de comunicación"*. También menciona que *"Las teorías etnomusicológicas"* afirman que *"la antigüedad del fenómeno musical en el hombre"... "nos remontan a hace unos 40.000 años"*, ya que en esa época *"comenzaron a perfilarse las primeras expresiones musicales asociadas a un hecho colectivo: rituales funerarios, cacerías y ceremonias vinculadas a la fertilidad"*, lo cual, en efecto, *"formaban parte de una cotidianidad de la que la música había entrado a formar parte por derecho propio."*. La música forma parte fundamental de la historia de la humanidad.

Por otro lado, no es fácil observar la utilidad de la música, pero pareciera ser un instrumento de encuentro en la humanidad y vinculado con la empatía que desarrollan las personas en sus relaciones interpersonales. Sin embargo, aunque el efecto de la música en nuestro cerebro es en gran parte un misterio, en varios casos descritos por el neurólogo Oliver Sacks, puede verse que la música afecta el desarrollo de la empatía en las personas [105].

La ***empatía*** está definida como el hecho de compartir e inferir las experiencias emocionales o sensitivas de otro [18] y tener respuestas emocionales adecuadas [56, 10]. En consecuencia, la empatía, como hecho humano, juega un rol crucial en las interacciones sociales y por ende en el modelo social en que vivimos. He ahí la importancia por comprender el comportamiento empático de las personas y cómo puede modificarse. En palabras de Mahatma Gandhi: *"Las tres cuartas partes de las miserias y malos entendidos en el mundo, terminarían si las personas se pusieran en los zapatos de sus adversarios y entendieran su punto de vista"*.

Habiendo dicho lo anterior, nace la problemática de cómo caracterizar la música de manera objetiva para poder realizar estudios científico-psicológicos de cómo afecta la música en la respuesta empática de las personas. Para esto, es necesario determinar qué características de la música podrían modular la empatía en el ser humano, y luego, extraer esas características de la música sistemáticamente.

A mayor abundamiento, es dable indicar que una característica que podría modular la empatía es la **emoción del sujeto**, ya que ésta determina la percepción y la actitud hacia el entorno, y en efecto, la música puede provocar cambios en la emoción que experimenta el oyente [74]. Es decir, la emoción que transmite la música es una característica relevante para aportar en un estudio relativo a la empatía.

Otra característica que posiblemente module la empatía es la **complejidad musical** de lo que se escucha, dependiendo dicha modulación no sólo de la complejidad, si no que también de la capacidad de comprensión musical del oyente. Para aclarar esta idea, imagine lo siguiente: un niño de 5 años escuchando música compleja, por ejemplo, un *Nocturne de Chopin*, posiblemente no comprenda lo que está escuchando y, por ende, su cerebro no reaccione al estímulo musical como lo haría el cerebro de un músico profesional. Así mismo, si el músico profesional escucha un tema infantil, como *"los pollitos dicen"*, su cerebro no se estimulará tanto como cuando escucha a Chopin. Es así que, dependiendo de la capacidad de comprensión musical del oyente, la complejidad de la música podría afectar en diferentes medidas a la respuesta cerebral hacia la música y eventualmente, a la respuesta empática de las personas.

En este punto se está en presencia de dos características de la música que podrían modular la respuesta empática de las personas: **1.** La emoción que inducen y **2.** Su complejidad. A continuación se abordará el tema de cómo extraer estas características de la música, de manera sistemática. Para esto es necesario definir cómo se representarán dichas características y como se obtendrán a partir de la música.

Las emociones han sido estudiadas extensamente en el ámbito de la psicología, siendo estas un elemento fundamental para la comprensión del ser humano. Pese a su complejidad existen diferentes enfoques para modelarlas. Russel (1980) propuso el **modelo circunflejo de las emociones** [1] donde las emociones se representan por su evaluación o valencia (*valence*) y su nivel de actividad o activación (*arousal*), de manera que cada emoción tiene un nivel de valencia y de activación. Estas variables conceptualmente pueden ir desde desagradable a agradable en el caso de la valencia y desde dormido hasta activado en el caso de la activación, existiendo combinaciones intermedias de ellas que pueden representar, por ejemplo; estrés, euforia, relajación o aburrimiento (ver figura 3.1).

Ahora bien, para mostrar parte del estado del arte de los esfuerzos que se han realizado por modelar las emociones que induce la música, es necesario presentar a **MediaEval**, una iniciativa que busca evaluar y comparar nuevos algoritmos útiles para la exploración y análisis de elementos de multimedia, enfocándose en los efectos sociales y humanos que éstos pueden generar. MediaEval invita a participantes a desarrollar algoritmos que logren cumplir los objetivos propuestos en cada prueba. En una versión de sus actividades solicitaron a los 21 equipos participantes generar un algoritmo que pudiese predecir de manera continua la emoción que induciría el extracto de una canción, entregándoles como punto de partida una base de datos (DEAM: *Database for Emotional Analysis of Music*) que contenía extractos de canciones, determinadas características de la onda sonora en el transcurso de cada canción, y las respectivas emociones representadas según el modelo circunflejo de las emociones de Russel en cada instante.

Pasamos a explicar el enfoque que se utiliza en este trabajo para lograr relacionar la música y su emoción, para lo cual es necesario mencionar lo siguiente: Para un humano es natural

determinar la emoción que le induce una música, pero diseñar un algoritmo determinista que lo haga es una tarea demasiado compleja. Encontrar analíticamente una función que tenga como entrada las características de la base de datos DEAM y como salida la emoción de la música sería una tarea casi imposible de realizar, ya que la onda sonora es muy compleja y no existe una clara relación con las emociones que provocan. Para este tipo de tareas, donde es difícil determinar una función que relacione ciertas variables de entrada con otras variables de salida, se puede utilizar la **Inteligencia Artificial** (AI del inglés *Artificial Intelligence*).

La AI es una disciplina que fue fundada académicamente en 1956 y que se refiere, en contraposición a la inteligencia natural presente en humanos y animales, a la inteligencia que pueden tener las máquinas, es decir, la característica que las hace capaces de resolver problemas y analizar datos externos sin que estén programadas explícitamente de manera determinista para hacerlo. Una de las ramas de la inteligencia artificial es el **aprendizaje de máquinas** (*Machine Learning*), el cual se ocupa de hacer que una máquina pueda aprender a generar una función basándose en datos de entrenamiento que es necesario proporcionarle. Dentro de esta categoría están las **redes neuronales artificiales** (ANN: *Artifitial Neural Network*), compuestas por nodos computacionales llamados **neuronas** o **perceptrones**, que imitan el comportamiento de una neurona cerebral (ver figura 2.8). Estos nodos se conectan entre sí activando sucesivos nodos que están en diferentes capas de nodos (ver figura 2.14). Cuándo estas capas son suficientes se está en presencia del **aprendizaje profundo** (*Deep Learning*).

Una ANN podría emular una función que tiene una entrada dada y se entrega su respectiva salida, pero cada ejemplo (entrada-salida) que se le entrega a la red es tomado como un ejemplo nuevo, sin tener en cuenta cual fue el ejemplo anterior, en otras palabras, una ANN, como la descrita hasta ahora, no tiene memoria.

Para intentar predecir la emoción que tiene una canción es necesario que el modelo que se utilice pueda "recordar" la canción desde su inicio mientras se le va entregando información de nuevos instantes de la canción. Este requerimiento tiene su base en que la música es un fenómeno temporal. Para entender intuitivamente el fundamento de dicho requerimiento se muestra lo siguiente: si un ser humano escucha un fragmento de 0.5 segundos de una canción, claramente esto no tendría ningún efecto sobre la emoción que experimenta, ya que la información contenida en esos 0.5 segundos carece de contexto. La solución a este problema son las **redes neuronales recurrentes** (RNN: *Recurrent Neural Network*), en las cuales la información es entregada de manera secuencial a una misma capa de neuronas y al momento de entrenarse lo hace teniendo en cuenta toda la secuencia entregada, ya que además de tener una entrada y una salida, tiene un estado, el cual conecta cada iteración en que la información atraviesa al bloque de la red. Estas redes también se consideran dentro del aprendizaje profundo puesto que la información pasa muchas veces por la misma capa. Lamentablemente las RNN tampoco son una opción adecuada, ya que durante su entrenamiento se puede producir **desvanecimiento o explosión del gradiente** (*vanish gradient o exploding gradient problem*), lo cual genera un aprendizaje deficiente cuando tiene que "recordar" una secuencia muy larga, dándole muy poca importancia (desvanecimiento) o demasiada importancia (explosión) a los últimos ejemplos de entrenamiento que se le entregan a la red.

Una red neuronal que se sobrepone a los problemas mencionados es la **red neuronal re-**

currente con memoria de corto y largo plazo (RNN-LSTM: *Recurrent Neural Network - Long-Short Term Memory*). En este tipo de red cada nodo es un bloque complejo que incluye estados de dicho bloque los cuales guardan la información del pasado y pueden transmitirla en cada iteración y, por ende, son adecuadas para ser entrenadas con información que posee sentido secuencial, es decir, cuando la información de entrenamiento es interpretable si se conocen no solo los ejemplos puntuales por separado si no que también la evolución de dichos ejemplos. Además, la estructura de estas redes permite seleccionar la información del pasado que tiene que seguir recordando"la red, mediante la utilización de compuertas especializadas en recordarü .ºlvidar", las cuales tienes sus propios parámetros, por lo que dicho tipo de red es capaz de aprender qué es lo que debe mantener en su memoria, superando así el problema del desvanecimiento o explosión del gradiente descendente.

Dados estos antecedentes se propone como candidato el algoritmo **RNN-LSTM** para caracterizar las emociones de la música basándose en las características de la onda.

En este punto es necesario volver atrás, donde se definieron las características de la música que se busca generar. Una de ellas es la emoción que induce la música, la cual será representada según el modelo circunflejo y la otra es la complejidad de la música. Dado que esta última podría evaluarse de diferentes maneras es que se implementan distintos algoritmos para generar diversas valoraciones de la complejidad. Dichos algoritmos tienen como entrada partituras de música representadas en formato ***kern***. Este formato es relativamente simple de procesar, ya que son archivos de texto plano donde los elementos de la partitura están representados de manera simbólica con combinaciones de caracteres ASCII, donde cada línea del archivo representa la aparición de notas en la partitura, es decir, avanzando de arriba hacia abajo en el archivo se leería la secuencia temporal de la melodía. Estos archivos son descargados de internet y el tipo de música que está en este formato se restringe a música docta de cámara, es decir, música generada con estándares académicos (docta) y que no tiene demasiados instrumentos (de cámara), a diferencia de la música orquestal. Se utilizan archivos *kern* de partituras de repertorios de diferentes compositores de los siglos XVII y XVIII.

Ahora bien, ¿qué se hace con estos archivos? Se utiliza la API (*Application Programming Interface*) ***Humdrum***, diseñada para trabajar con este tipo de archivos, con el propósito de transformar la concatenación de todas las canciones de cada repertorio, en diferentes parámetros, los que sirven como punto de partida para obtener la complejidad que tienen los repertorios de cada compositor. Estos parámetros son secuencias de números enteros generadas a partir de la secuencia de información que contienen los *kern* línea tras línea. Cada uno de los parámetros, obtenidos de cada uno de los repertorios, pueden considerarse como un mensaje desde el punto de vista de la **teoría de la información** (TI) y en efecto, se caracteriza cada parámetro aplicando medidas definidas en el marco de la TI.

La TI, propuesta por Claude Shannon en 1948 [114] plantea una forma de comprender y analizar la información basándose en la *teoría de las probabilidades* y en la descripción de la información según las *ciencias de la computación*, generando conceptos y formulaciones matemáticas que han aportado al desarrollo en las tecnologías de las comunicaciones y de los medios digitales, principalmente optimizando la transmisión y almacenamiento de información.

Uno de los conceptos fundamentales propuestos en la TI es la ***entropía***, elemento con que se representa la complejidad en este trabajo. La entropía cuantifica la complejidad de un sistema de información, o dicho de otra manera, la incertidumbre que se tiene acerca de qué elementos tendrá un mensaje. Como se mencionó, los parámetros obtenidos de los archivos *kern* se consideran mensajes, a los cuales, en efecto, se les aplicará la función de la entropía para así obtener diferentes representaciones de la complejidad. Además se obtendrán sus entropías condicionales y entropías normalizadas, funciones que utilizan el concepto de la entropía, generando otros caminos para obtener una mayor variedad de representaciones de la complejidad. De esta forma, se implementarán diferentes algoritmos -compuestos por diferentes medidas entrópicas aplicadas a diferentes parámetros secuenciales- a cada uno de los repertorios para así poder comparar las combinaciones de algoritmos *entrópico-paramétricos* y seleccionar el que represente la complejidad de mejor manera.

Recapitulando lo expuesto hasta ahora, se han descrito superficialmente como se obtienen representaciones de la emoción que induce la música en el transcurso del tiempo y de la complejidad que tienen los repertorios de distintos compositores. Ahora bien, ¿cómo se podría utilizar esto para investigar cómo afectan dichas características de la música en la respuesta empática de las personas? Claramente esta es una pregunta que escapa al campo de la ingeniería, a pesar de ello, en este trabajo se propone un diseño experimental que eventualmente respondería a dicha pregunta.

El diseño experimental que se propone, descrito someramente, consiste en registrar la actividad cerebral de un individuo utilizando EEG, mientras ve imágenes con y sin contenido de dolor en forma secuencial y escucha diferentes extractos de canciones, caracterizados según la emoción que inducen o según su complejidad; o simplemente ve las imágenes sin estímulo sonoro (fase de control). Además el sujeto será sometido a diversos cuestionarios para complementar la información de la actividad cerebral. Con los datos recabados, eventualmente se podría encontrar correlaciones entre la respuesta empática que tienen los sujetos, medida por la supresión de la onda mu registrada por el EEG, y los valores de las características de la música que escucharon. Este apartado tiene como objetivo introducir un uso práctico que podrían tener las herramientas ingenieriles diseñadas e implementadas.

A pesar de que la comprensión de la relación entre la música y la empatía es el trasfondo que motiva este trabajo, el diseño del experimento no forma parte de los objetivos fundamentales de esta memoria, los que, en efecto, son diseñar e implementar los instrumentos ingenieriles descritos en los párrafos anteriores. Esto no quiere decir que la propuesta del diseño experimental sea insuficiente o sus cimientos sean frágiles, de hecho, se realizó una extensa revisión bibliográfica para la contextualización del tema en cuestión, y que tiene como producto el diseño en detalle de un experimento, fundamentado apropiadamente. Ahora bien, es necesario separar el tema ingenieril de lo relacionado al experimento, lo cual se logra con una organización adecuada de este documento, que se describe en el último párrafo de esta introducción.

En resumen, este trabajo busca generar 2 instrumentos ingenieriles: uno que sirva para obtener la emoción que induce la música sobre las personas, basándose en características físicas de la onda sonora, y otro que arroje la complejidad que tienen repertorios de diferentes compositores, basándose en la información que contienen sus partituras. Esto con el objeto

de aportar a la investigación de cómo afecta la música en la empatía de las personas, para lo cual, además, se ha diseñado un experimento que eventualmente serviría para determinar la correlación entre las características de la música, tanto de emoción como de complejidad, y su efecto sobre la empatía, medido por la supresión de la señal de la onda mu. La correlación en particular que se usaría no se define en este trabajo. El diagrama general de lo que se busca lograr, incluyendo el experimento propuesto, se muestra en la figura 1.

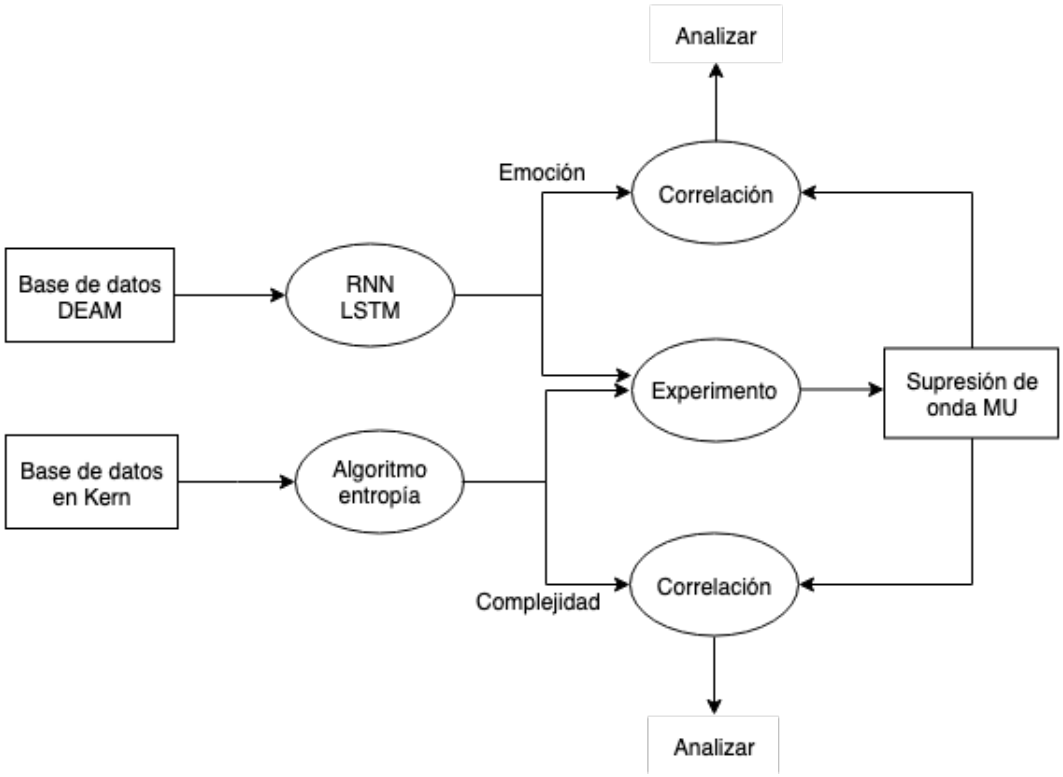

Figura 1: Flujo general simplificado.

En las páginas siguientes se presenta el cuerpo de este trabajo, el cual está estructurado de la siguiente manera: en el *capítulo 1* se presentan los objetivos y alcances de este trabajo; en los *capítulos 2, 3* y *4* las bases teóricas necesarias respecto al aprendizaje de máquinas y a la teoría de la información, que contemplaran las definiciones, conceptos, ecuaciones y herramientas ingenieriles utilizada para las caracterizaciones de la música; seguido a esto, en el *capítulo 5*, se introducen las herramientas computacionales utilizadas; luego, en el *capitulo 6*, se detallan los procedimientos y metodologías utilizadas, relativas al aprendizaje de máquinas y a la teoría de la información, con sus respectivas argumentaciones; el *capítulo 7* incluirá los resultados del trabajo de caracterización junto al análisis correspondiente de los mismos; en el *capítulo 8* se desarrollará todo lo referente al experimento, comenzando por describir el estado del arte en cuanto a los estudios que involucran el cerebro y la empatía, para luego describir el diseño experimental propuesto; y, finalmente, el *capítulo 9* indicará las conclusiones de este trabajo y se plantearán posibles cambios que podrían mejorar los resultados, además de direcciones que se podrían tomar en la continuación de este. En los anexos (A y B) se podrán examinar los códigos utilizados y resultados considerados redundantes para el análisis realizado, respectivamente.

Capítulo 1

Objetivos y Alcances

1.1. Objetivos

1.1.1. Objetivo General:

Diseñar e implementar programas para caracterizar música según la emoción que provocan y según su entropía, para aportar estas herramientas, junto con un diseño experimental adecuado, a la investigación sobre como afecta la música en la empatía.

1.1.2. Objetivos Específicos:

- Caracterizar música según la emoción que inducen en cada instante, según modelo circunflejo de las emociones, utilizando RNN-LSTM.
- Obtener la entropía de primer orden, entropía normalizada y entropía condicional de diferentes parámetros de ciertos repertorios de compositores de música de cámara.
- Caracterizar dichos repertorios según la medida entrópica aplicada al parámetro que mejor clasifique los estilos de los compositores según su complejidad melódica.
- Diseñar un experimento que utilice las características generadas y sirva para determinar la relación entre dichas características y su efecto sobre la empatía.

1.2. Alcances

- Se obtiene la entropía de música de cámara que está en formato *kern* de diferentes compositores. No se obtiene la entropía de música en otros formatos. Tampoco se generan los archivos *kern*, si no que se descargan de internet.

- El diseño de la red neuronal y del preprocesamiento de los datos se realiza utilizando métodos ya existentes, no se crearán nuevos métodos, por lo que la innovación en la propuesta radica en la combinación de los métodos que se aplican. No se pretende mejorar el estado del arte en cuanto a los resultados de modelos ya implementados.

- Se diseña un experimento piloto que utiliza EEG y encuestas psicológicas, quedando como trabajo futuro la implementación de este.

Capítulo 2

Aprendizaje de Máquinas

En esta sección se presentan las herramientas necesarias para poder explicar en el capítulo 6 cómo se diseñó e implementó el algoritmo de aprendizaje de máquinas para la obtención de las emociones. Este capítulo está ordenado de manera orgánica, dando en primer lugar una mirada amplia del tema, para luego describir las herramientas que se utilizan efectivamente en este trabajo.

El interés de la humanidad por comprender la lógica y poder desarrollar algoritmos y máquinas que la utilicen, data de hace mucho tiempo. Poder generar máquinas capaces de pensar para que nos ayuden a resolver problemas y analizar datos es hoy en día un tópico fundamental para el desarrollo de la tecnología, que nos está llevando a avances insospechados en el tiempo en que no existían computadores. Estos avances, de hecho, han expandido los horizontes de la civilización, llegando a concebir y a crear otra inteligencia, creada por la nuestra, pero con consecuencias muy difíciles de dimensionar. Este importante desarrollo puede centrarse en torno a un concepto: la inteligencia artificial, en adelante AI (*Artifitial Intelligence*).

La AI es la disciplina que, a grandes rasgos, busca generar algoritmos que sean capaces de aprender a resolver problemas, sin que se les indique explícitamente los pasos a seguir para resolver dicho problema, si no que dándole una serie de *reglas*, con las cuales el algoritmo puede evolucionar y *aprender* por si mismo. La versión más simple de AI es cuando dichas reglas son diseñadas manualmente (*Rule-based system*). Dentro de la AI se encuentra el aprendizaje de máquinas (ML, del inglés *Machine Learning*), el cual consiste en crear máquinas que sean capaces de adquirir su propio conocimiento, extrayendo patrones de los datos. Para esto puede ser necesario representar los datos de una manera en que sea útil para el propósito de aprendizaje. Cuando se está en presencia de un algoritmo de ML, el cual utiliza un cambio en la representación de los datos para aprender lo que necesita aprender, entonces se está en presencia del aprendizaje de representación (RL, del ingles *Representation Learing*). Para finalizar esta cadena de sub-disciplinas de la AI se presenta el aprendizaje profundo (DL, de *Deep Learning*), el cual forma parte del RL, pero en particular genera varias transformaciones secuenciales a los datos, para así obtener representaciones de variadas complejidades según que tan profundo se esté dentro del algoritmo en cuestión, es decir, que tantas funciones se le hayan aplicado a los datos; y luego, en base a estas representaciones, obtener el resultado

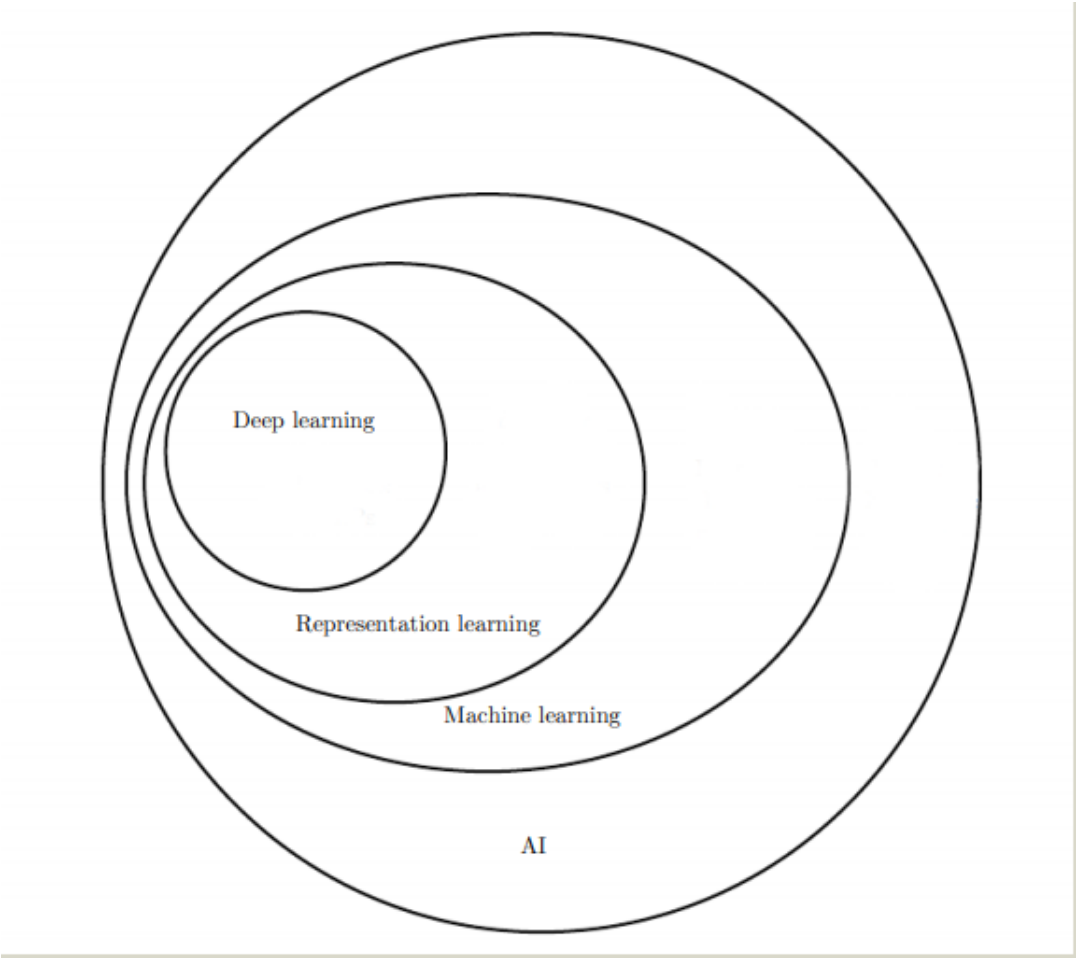

Figura 2.1: Diagrama de Venn de las Sub-disciplinas de la inteligencia artificial (adaptada de [51]).

que se está buscando.

En otras palabras, la AI es una disciplina que busca generar funciones que se adapten a ciertos modelos. El ML genera esa función en particular *aprendiendo* de datos que le son entregados. Los algoritmos de RL toman los datos y los transforman de alguna forma antes de obtener el resultado final de la función, mientras que los modelos de DL transforman los datos muchas veces siendo la función generada por este tipo de algoritmos una composición de varias funciones. La relación de pertenencia de las disciplinas presentadas puede verse en la figura 2.1 y el flujo que toma la información en los diferentes paradigmas puede verse en la figura 2.2.

Para entender más a fondo estas disciplinas es necesario definir algunos elementos existentes en ellas, lo que se hará comenzando desde el panorama amplio de un algoritmo cualquiera perteneciente a ellas y recorriendo sus partes y procesos necesarios. Se restringirá al subconjunto de algoritmos de ML, los cuales, como fue mencionado, aprenden de los datos.

Primero es necesario definir el algoritmo a utilizar e identificar los **hiper-parámetros** que lo definen, los cuales juegan un rol crucial en el desempeño del algoritmo. La naturaleza de

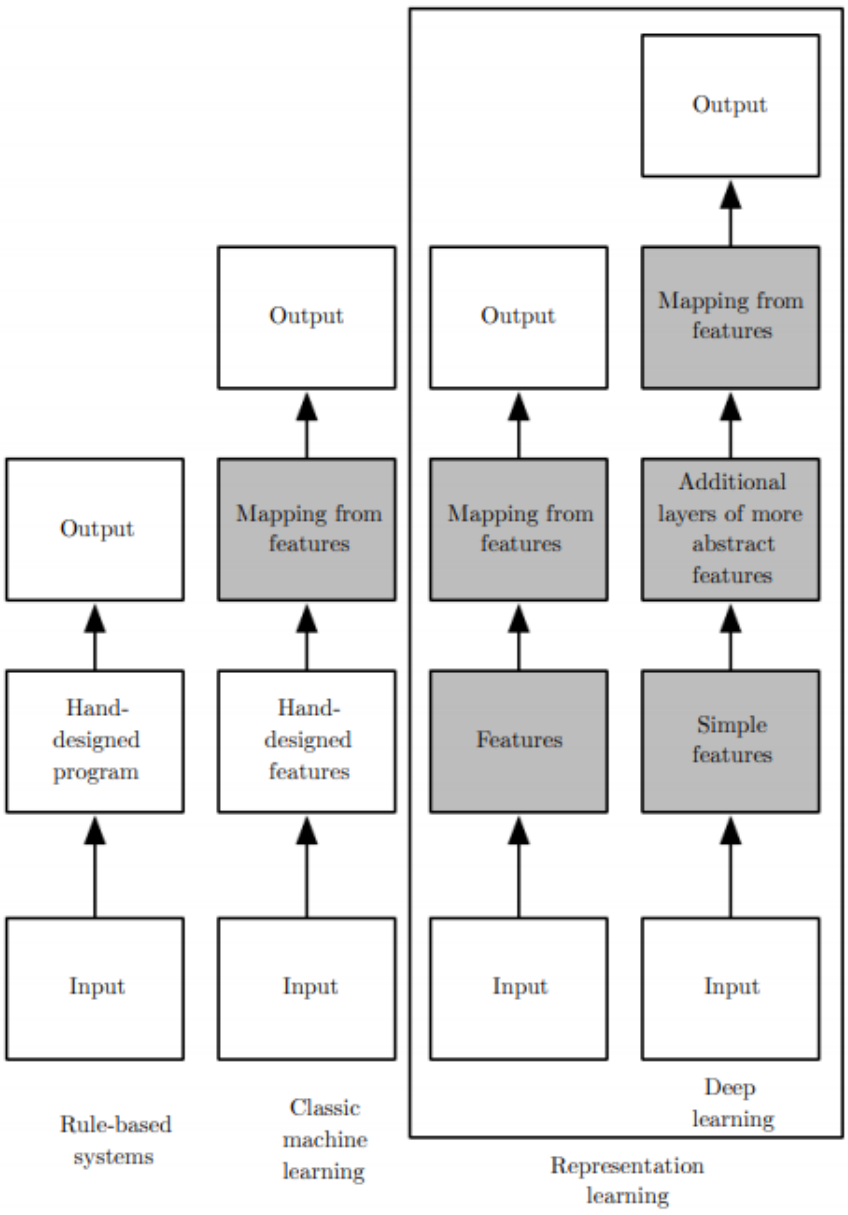

Figura 2.2: Diagrama de Flujo de la información en las diferentes sub-disciplinas de la AI (adaptada de [51]).

los hiper-parámetros depende del algoritmo en particular, por lo que serán abordados en la sección correspondiente a cada algoritmo a implementar, pero se dirá por ahora que definen la estructura y dinámica de aprendizaje del algoritmo. La implementación de estos algoritmos, aparte del diseño de la estructura de éste, puede dividirse básicamente en dos etapas. Primero una de **entrenamiento**, dónde el algoritmo aprende, es decir, dónde va modificando iterativamente sus **parámetros**, los cuales definen la función específica que está modelando el algoritmo, y luego una etapa de **test**, dónde se comprueba si el algoritmo efectivamente aprendió. Para poder llevar acabo estos procesos, como se dijo, es necesario disponer de datos, a cada uno de estos datos se les denomina **ejemplos** y puede estar representado de múltiples maneras, generalmente por vectores donde cada dimensión del vector representa una **característica**. Previo a entrenar y testear el algoritmo es necesario explorar los datos y preprocesarlos adecuadamente para entregárselos al algoritmo de aprendizaje, el cual se debe escoger según la naturaleza del problema a resolver y de los datos que se disponen.

Luego de preprocesar los datos es necesario separarlos en un **conjunto de entrenamiento** y un **conjunto de test**. Estos conjuntos son disjuntos y se toma alrededor del 80 o 90 % de todos los ejemplos para el conjunto de entrenamiento y el resto para el conjunto de test.

En la fase de entrenamiento el algoritmo va recibiendo los ejemplos del conjunto de entrenamiento uno a uno y va modificando sus parámetros con el objetivo de minimizar cierta función objetivo que se define basado en el problema que se está intentando solucionar. Cuando el algoritmo termina se pasa a la etapa de test. En esta etapa no se modifican los parámetros y se utilizan los ejemplos pertenecientes al conjunto de test, para observar como se comporta el algoritmo entrenado con datos que no conoce.

Mientras que en la etapa de entrenamiento se tiene un error asociado a la función objetivo a minimizar, denominado **error de entrenamiento**, en la etapa de test se tendrá el respectivo **error de test** o llamado también **error de generalización**. Lo que se busca al entrenar un algoritmo de ML es obtener el menor error de test, para lo cual es necesario obtener un bajo error de entrenamiento. La potencialidad que tiene un algoritmo de obtener un bajo error de entrenamiento se denomina **capacidad** y se busca que ésta sea lo suficientemente alta para obtener un error de entrenamiento bajo. Si la capacidad del algoritmo es demasiado alta, y por ende su error de entrenamiento es bajo, pero al mismo tiempo el error de test es alto en comparación al de entrenamiento, entonces se estaría en presencia de un **sobre-ajuste** (*overfitting*). En el otro caso, cuando el error de entrenamiento no alcanza el rango esperado se está en presencia de un **sub-ajuste** (*underfitting*). Ni uno de estos dos casos es lo esperado, lo ideal en cambio es que la capacidad esté en un punto intermedio dónde el error de entrenamiento sea bajo y el error de prueba sea lo más similar posible al error de entrenamiento, es decir, se busca minimizar la brecha entre el error de test y el error de entrenamiento. Estas 3 situaciones pueden observarse en la figura 2.3. Cuando se está en la situación adecuada se dice que existe **generalización**, ya que el algoritmo muestra que su aprendizaje puede generalizarse a datos que aún no conoce. Cuando se está en la zona de sobre-ajuste es necesario realizar ciertos cambios al modelo para que logre generalizar adecuadamente. El conjunto de estrategias utilizadas para este fin se conoce como **regularización** y es explicada con mayor detalle en el apartado 2.3.7.

Para poder seleccionar los hiper-parámetros se necesita otro conjunto para ir testeando

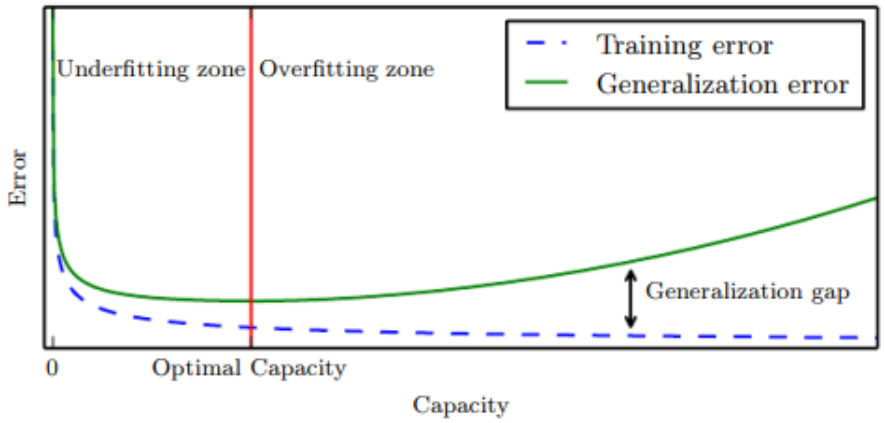

Figura 2.3: Capacidad v/s Errores: Gráfico del comportamiento del error de test y del error de entrenamiento en función de la capacidad del algoritmo (adaptada de [51]). Puede verse que en la zona izquierda hay sub-ajuste (*underfitting*), mientras que en la zona derecha se está en presencia de sobre-ajuste (*overfitting*). Lo ideal es diseñar un algoritmo que se encuentre en la línea roja, dónde se logra generalizar adecuadamente.

los resultados con los diferentes hiper-parámetros que se le entrega a la red, este conjunto se denomina **conjunto de validación** al cual se asocia el proceso de validación. Este conjunto es una parte del conjunto de entrenamiento, siendo lo restante denominado también conjunto de entrenamiento. Entonces se tienen 3 conjuntos finales: entrenamiento, test y validación (ver figura 2.24), junto a las 3 etapas respectivas (ver figura 2.4), dónde sólo en el entrenamiento se entrenarán los parámetros y en entrenamiento y validación se probarán con diferentes hiper-parámetros, para finalmente, en el test, obtener el desempeño del algoritmo generado .

En resumen, desde un punto de vista general, para lograr un algoritmo de ML, es necesario obtener los datos, procesarlos adecuadamente, escoger un modelo, entrenarlo, testearlo, e ir mejorando dicho modelo hasta lograr el objetivo. Dicho procedimiento se muestra en la figura 2.5.

Yendo un poco más allá en la comprensión de los algoritmos de ML es necesario mostrar diferentes formas de clasificarlos, y así expandir el panorama actual. El aprendizaje de un algoritmo de ML puede ser **supervisado** o **no supervisado**. Se considerarán las siguientes definiciones para estos conceptos: El aprendizaje supervisado es aquel en que los datos que se le dan a la máquina para su entrenamiento vienen pareados, donde el primer elemento del par corresponde a un vector de características de entrada (*features*), mientras que el segundo es un vector que representa la salida objetivo (*target*), es decir, la salida que debiese arrojar el algoritmo cuando se le entrega el respectivo vector de características de entrada. Por otro lado, el aprendizaje no supervisado es el en que solo se le dan las características de entrada, lo cual puede servir para la agrupación de los datos (*clustering*), con el fin de observarlos, o para generar nuevas representaciones de estos utilizando autocodificadores (*auto-encoders*).

Otra forma de clasificar los algoritmos inteligentes, es según como es el dominio de los datos objetivos, es decir, que tipo de datos se necesita predecir. En relación a esto están los algoritmos de **clasificación** y los algoritmos de **regresión**. Al entrenar una red se pueden

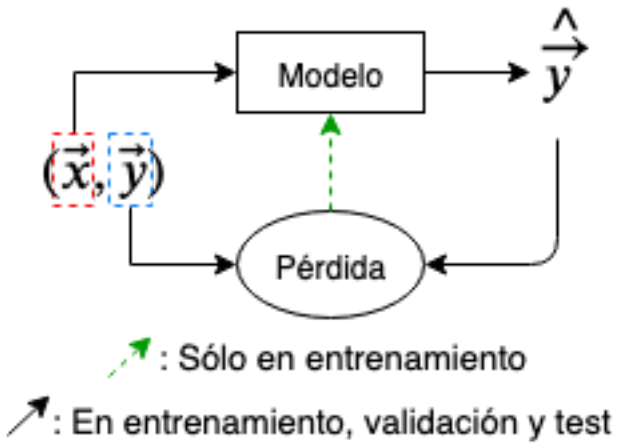

Figura 2.4: Flujo en entrenamiento, validación y test.

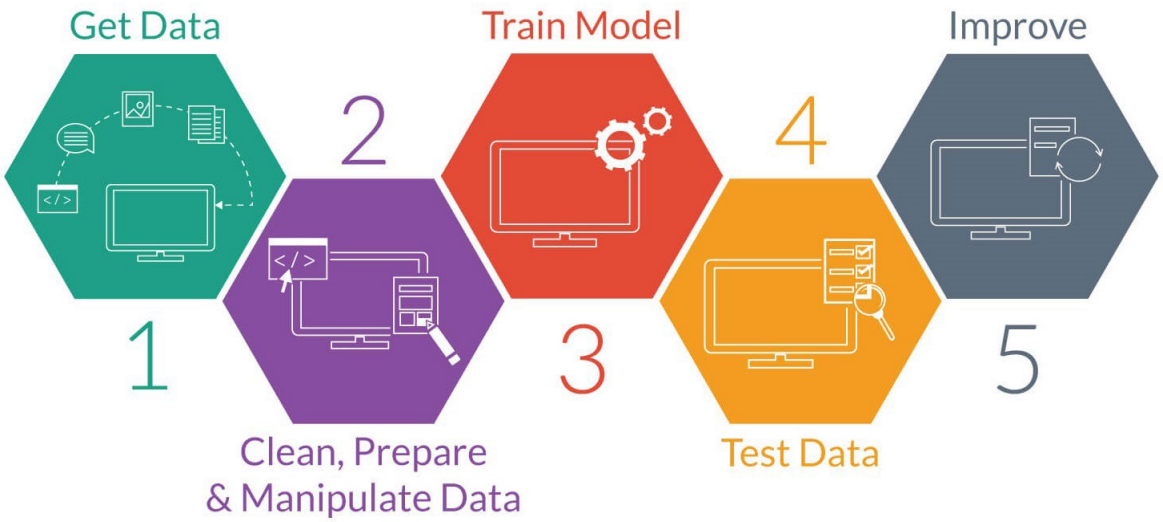

Figura 2.5: Representación general de los procesos fundamentales en la implementación de un algoritmo de ML (adaptada de [39]).

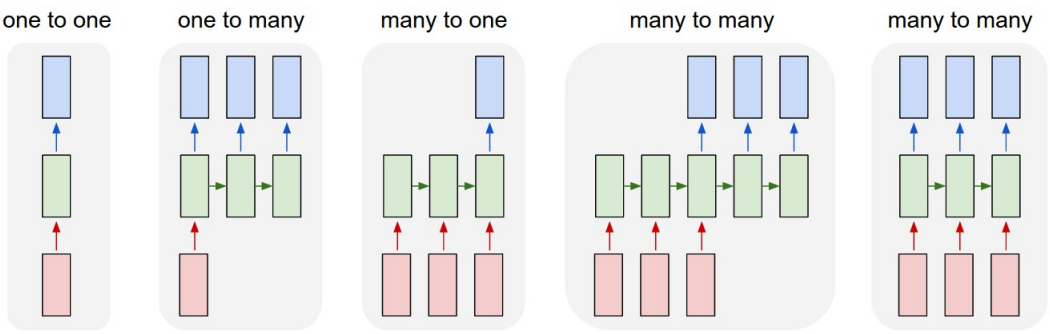

Figura 2.6: Posibles relaciones entre largos de secuencias de datos de entrada y de salida de una RNN (adaptada de [64]).

tener variables de salida discretas o continuas, por ejemplo, si dados los valores de los pixeles de la imagen de un rostro humano, se quisiera determinar la emoción que está sintiendo, que podría ser tristeza, felicidad, enojo, etc., entonces estaríamos en presencia de salidas discreta, y se representaría cada emoción con un número en particular. En este caso se habla de clasificación. En otra situación, se podría querer determinar qué tan feliz y enojada, en un rango de 1 a 10, continuo, está la persona de la imagen, en cuyo caso se tendría dos elementos en el vector de salida, felicidad y enojo, y se estaría en presencia de una regresión.

Una última forma de clasificación, importante de mencionar para este trabajo, dónde se utilizan secuencias de datos y por ende redes neuronales recurrentes (ver 2.3.4), es la forma en que se relacionan los datos de entrada y los datos de salida. Estas pueden ser las siguientes: **1.** Que un elemento de entrada produzca un elemento de salida (***one-to-one***). **2.** Que un elemento de entrada produzca una secuencia de elementos de salida (***one-to-many***). **3.** Que una secuencia de elementos de entrada produzca un elemento de salida (***many-to-one***). **4.** Que una secuencia de elementos de entrada produzca una secuencia de elementos de salida, donde cada elemento de entrada produce un elemento de salida en cada iteración (***many-to-many***). Y **5.** Que una secuencia de elementos de entrada produzca una secuencia de elementos de salida luego de que haya terminado de ingresar la secuencia de entrada a la red (***many-to-many***). Lo anterior se muestra gráficamente en la figura 2.6.

Por último, antes de empezar a describir los métodos utilizados en sí, es importante mencionar que el proceso de optimización implicado en el entrenamiento de los algoritmos tiene características diferentes a la optimización de una función cualquiera, ya que el objetivo de la optimización en ML es minimizar el error de test, pero se hace minimizando el error de entrenamiento, por lo que es una optimización indirecta. Se describirá más en profundidad como funciona la optimización en ML y diferentes enfoques en el apartado 2.3.6, al igual como se hace con la regularización en 2.3.7, ya que son dos temas fundamentales en cualquier algoritmo de ML.

2.1. Preprocesamiento de los datos

En esta sección se explican las dos herramientas estadísticas que se utilizan para discriminar entre los datos: La **Correlación de Pearson** y el **Alfa de Cronbach**

2.1.1. Correlación de Pearson

Relación entre las dispersiones de 2 variables x e y, definidas por la ecuación 2.1.

$$r = \frac{\sum_{\mathrm{i}=1}^{n}(x_{\mathrm{i}} - \bar{x})(y_{\mathrm{i}} - \bar{y})}{\sqrt{\sum_{\mathrm{i}=1}^{n}(x_{\mathrm{i}} - \bar{x})^2}\sqrt{\sum_{\mathrm{i}=1}^{n}(y_{\mathrm{i}} - \bar{y})^2}} \tag{2.1}$$

Dónde $\bar{\boldsymbol{x}}$ e $\bar{\boldsymbol{y}}$ son los promedios de las variables x e y respectivamente.

La correlación de Pearson puede expresarse en términos más reducidos (ec. 2.2:

$$r = \frac{\sigma_{x,y}}{\sigma_x \sigma_y} \tag{2.2}$$

Dónde:
$\boldsymbol{\sigma_{x,y}}$ **:** Covarianza entre las variables x e y.
$\boldsymbol{\sigma_x}$ **:** Desviación Estándar de la variable x, idem y.

2.1.2. Alfa de Cronbach

Representa fiabilidad, si $0{,}7 < \alpha < 0{,}9$, la muestra es confiable [126]. Se puede obtener mediante la varianza de los ítems (ec. 2.3) o mediante la matriz de correlación (ec. 2.4), que usa las correlaciones de Pearson.

$$\alpha = \frac{k}{k-1}\left[1 - \frac{\sum v_{\mathrm{i}}}{v_t}\right] \tag{2.3}$$

Dónde:
$\boldsymbol{\alpha}$ **:** Alfa de Cronbach
$\boldsymbol{k}$ **:** Número de ítems
$\boldsymbol{v_{\mathrm{i}}}$ **:** Varianza de cada ítem.
$\boldsymbol{v_t}$ **:** Varianza del total

$$\alpha_{est} = \frac{kp}{1 + p(k-1)} \tag{2.4}$$

Dónde:
$\boldsymbol{p}$ **:** Promedio de correlaciones entre cada uno de los ítems.

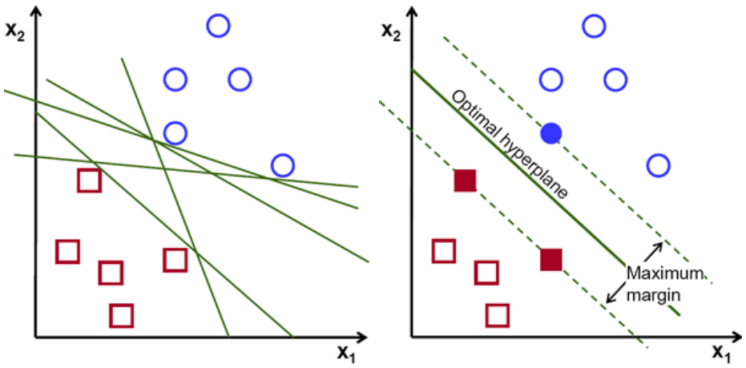

Figura 2.7: Optimización SVM (adaptada de [127]).

2.2. Máquina de Vectores de Soporte (SVM)

Una Máquina de Vectores de Soporte (SVM de sus siglas en inglés: *Support Vector Machine*) es un conjunto de algoritmos utilizados para entrenar un modelo que sea capaz de clasificar un conjunto de datos en diferentes clases (SVC - *Support Vector Classification*) o realizar una regresión a dichos datos (SVR - *Support Vector Regression*). Se extiende a continuación la explicación teórica de SVC.

Su funcionamiento se basa en la existencia de un hiperplano que podría separar los elementos en 2 clases. Ya que pueden existir infinitos hiperplanos que logren esta separación, es que SVC busca al mejor, es decir, al hiperplano que maximice los márgenes que genera, siendo los márgenes las distancias entre el hiperplano y las rectas paralelas a éste, que pasan por el (los) elemento(s) de cada clase más cercano al hiperplano. Estos elementos se denominan **vectores de soporte** (*support vectors*) y en la figura 2.7 se representan por el círculo y los cuadrados rellenos del gráfico de la derecha de dicha figura.

Matemáticamente, encontrar un hiperplano es similar a encontrar una recta, pero en vez de ser representado por $y = m \cdot x + n$, donde cada elemento es un escalar, se representa según la ecuación 2.5:

$$f(\vec{x}) = \vec{w}^T \cdot \vec{x} + b \tag{2.5}$$

Dónde:
$\vec{x}$: Vector de características a separar.
$\vec{w}$: Vector de pesos de cada dimensión.
b : Umbral.
$f(x)$: Función de decisión.

Es decir, se utilizan vectores, cuya dimensión es la cantidad de características que tienen los elementos a separar (n) y $f(x)$ definirá a qué clase corresponde el vector $\vec{x}$ [43]. Las clases generadas por este algoritmo son 2, una cuando $f(x)$ es mayor o igual que 1 y otra para cuando $f(x)$ menor o igual que -1 como puede verse en la ecuación 2.6, dónde las clases se definen como $y = 1$ e $y = -1$, respectivamente para cada caso de $f(x)$. Luego, se puede

reescribir la función de decisión, incluyendo la clase a la cual pertenecen como se muestra en la ecuación 2.7.

$$\begin{array}{ll} \vec{w}^T \cdot \vec{x}^{(\mathrm{i})} + b \geq 1 & si\ y^{(\mathrm{i})} = 1 \\ \vec{w}^T \cdot \vec{x}^{(\mathrm{i})} + b \leq 1 & si\ y^{(\mathrm{i})} = -1 \end{array} \tag{2.6}$$

$$y^{(\mathrm{i})} \cdot (\vec{w}^T \cdot \vec{x}^{(\mathrm{i})} + b) \geq 1 \tag{2.7}$$

Un elemento clave que se puede usar en combinación con SVM es el **truco del kernel** (*kernel trick*), el cual se basa en el hecho de que muchos algoritmos de ML pueden ser reescritos en términos de los productos puntos entre los ejemplos. En este caso, la ecuación 2.5 puede reescribirse de la siguiente manera (ec. 2.8):

$$f(\vec{x}) = \sum_{\mathrm{i}=1}^{m} \vec{\alpha}_{\mathrm{i}} \vec{x}^T \cdot \vec{x}^{(\mathrm{i})} + b \tag{2.8}$$

Dónde $\vec{x}^{(\mathrm{i})}$ corresponde a un ejemplo y $\vec{\alpha}_{\mathrm{i}}$ a un vector de coeficientes. Lo anterior posibilita transformar los vectores $\vec{x}$ y $\vec{x}^{(\mathrm{i})}$ en sus respectivas transformaciones $\phi(\vec{x})$ y $\phi(\vec{x}^{(\mathrm{i})})$, las que tienen como resultado vectores de mayor dimensionalidad que los de 2.8 y por ende permiten realizar separaciones más complejas de los datos. Al multiplicar usando producto punto entre $\phi(\vec{x})$ y $\phi(\vec{x}^{(\mathrm{i})})$ resulta el **kernel** (ec. 2.9).

$$K(\vec{x}, \vec{x}^{(\mathrm{i})}) = \phi(\vec{x}) \cdot \phi(\vec{x}^{(\mathrm{i})}) \tag{2.9}$$

Al hacer la transformación anterior, reemplazando el kernel por el producto de los vectores de la ecuación 2.8, resulta lo que se muestra en la ecuación 2.10. Ahora $f(\vec{x})$ deja de ser lineal con respecto a $\vec{x}$, pero si es lineal con respecto a $\vec{\alpha}_{\mathrm{i}}$, lo cual extiende las posibilidades de SVC a poder separar conjuntos de datos que no son linealmente separables y permite usar optimización convexa debido a la linealidad que tiene con respecto a la variable de optimización $\vec{\alpha}_{\mathrm{i}}$. Uno de los kernels más usadas es la **función de base radial** (RBF de *radial basis function*), definida por la ecuación 2.11:

$$f(\vec{x}) = \sum_{\mathrm{i}=1}^{m} \vec{\alpha}_{\mathrm{i}} K(\vec{x}, \vec{x}^{(\mathrm{i})}) + b \tag{2.10}$$

$$K(\vec{x}, \vec{x}^{(\mathrm{i})}) = \mathrm{e}^{-\frac{\|\vec{x} - \vec{x}^{(\mathrm{i})}\|}{2\sigma^2}} \tag{2.11}$$

La ecuación 2.11 se puede reescribir utilizando el parámetro $\gamma = \frac{1}{2\sigma^2}$, resultando en la ecuación 2.12.

$$K(\vec{x}, \vec{x}^{(\mathrm{i})}) = \mathrm{e}^{-\sigma\|\vec{x} - \vec{x}^{(\mathrm{i})}\|} \tag{2.12}$$

El hiper-parámetro γ representa que tan lejos llega la influencia de un ejemplo en particular, valores bajos de γ significa que su influencia llega lejos, mientras que valores alto lo contrario. Este hiper-parámetro se considera en la optimización de la posición del hiperplano.

Para maximizar la distancia entre los vectores de soporte y el hiperplano primero se define la distancia según la ecuación 2.13.

$$\rho(\vec{w}, b) = \min_{\{\vec{w},\vec{x}:y=1\}} \frac{\vec{x} \cdot \vec{w}}{|\vec{w}|} - \max_{\{\vec{w},\vec{x}:y=-1\}} \frac{\vec{x} \cdot \vec{w}}{|\vec{w}|} \tag{2.13}$$

Y se puede demostrar que maximizar la ecuación 2.13 es equivalente a maximizar la ecuación 2.14, lo que a su vez se logra minimizando $\vec{w}^T \cdot \vec{w}$.

$$\rho(\vec{w}_0, b_0) = \frac{2}{|\vec{w}_0|} = \frac{2}{\sqrt{\vec{w}_0 \cdot \vec{w}_0}} \tag{2.14}$$

Luego, lo que se busca es minimizar la multiplicación de los pesos, sujeto a la condición de clasificación (ec. 2.7).

Hasta el momento se tiene un algoritmo capaz de clasificar datos que no son linealmente separables aplicando un kernel, pero muchos conjuntos de datos de la vida real, luego de aplicar un kernel, tampoco es posible separar. Esto puede deberse a la presencia de *outliers* o a la naturaleza de los datos. Dado esto se genera una variación del algoritmo que incluye hiperplanos de márgenes suaves (*soft margin hyperplane*). En esta versión se admiten errores, pero se quieren minimizar, por lo que finalmente lo que se busca es minimizar la multiplicación de los pesos más una ponderación de los errores, sujeto a que se cumpla la condición de clasificación. Este resultado final puede verse en la ecuación 2.15

$$\begin{gathered} \min_{\vec{w},b,\xi} \frac{1}{2} \cdot \vec{w}^T \cdot \vec{w} + C \sum_{\mathrm{i}=1}^{l} \xi_{\mathrm{i}} \\ s.\,a.\ y^{(\mathrm{i})} \cdot (\vec{w}^T \cdot \vec{x}^{(\mathrm{i})} + b) \geq 1 - \xi_{\mathrm{i}} \\ C > 0,\ \xi_{\mathrm{i}} \geq 0 \end{gathered} \tag{2.15}$$

Dónde C es otro hiper-parámetro, el cuál es un *trade-off* entre la correcta clasificación de los ejemplos de entrenamiento y el margen de la función de decisión. Un valor de C bajo fomentará un margen más grande y por ende una función de decisión más simple, disminuyendo la precisión en el entrenamiento. ξ_{i} es el error en cada ejemplo.

En resumen, SVC primero transforma los datos a un espacio de mayor dimensionalidad dónde sea factible separarlos y luego optimiza la posición del hiperplano separador par que este separe las clases adecuadamente y dicho hiperplano esté lo más distante posible a los vectores de soporte.

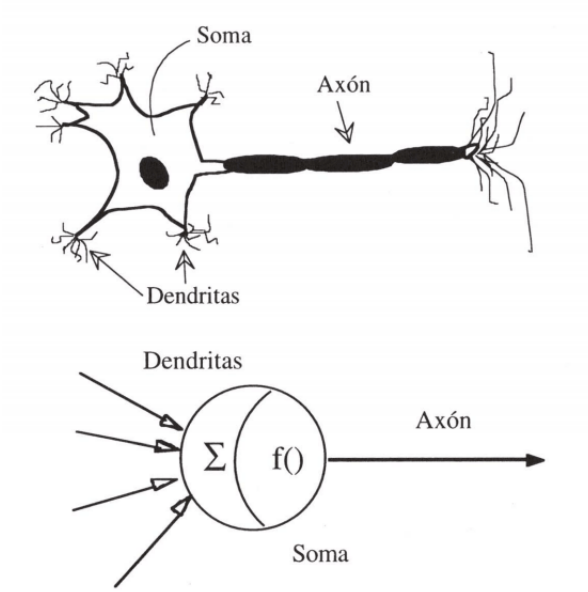

Figura 2.8: Comparación estructural entre neurona biológica y neurona artificial (adaptada de [37]).

2.3. Redes Neuronales Artificiales (ANN)

Las redes neuronales artificiales(ANN del inglés *Artificial Neural Network*), o simplemente redes neuronales (NN), son un tipo de algoritmo de ML, que fue inspirado por la actividad bioeléctrica que se produce en el cerebro. La estructura de estos algoritmos consiste en la conexión de **neuronas artificiales** o **nodos**, que imitan el comportamiento de las neuronas cerebrales, visto desde un punto de vista simplificado. Existen varias analogías entre ambos sistemas. Las neuronas cerebrales conducen electricidad, mientras que las neuronas artificiales conducen información en forma de números reales. Las primeras se unen entre ellas para generar redes neuronales, al igual que las neuronas artificiales generan redes neuronales artificiales al unirse. Las redes neuronales cerebrales son entrenadas y producen las respuestas que genera nuestro organismo dados los estímulos existentes. Las NN también son entrenadas y dadas ciertas entradas, generan ciertas salidas. En el caso de las neuronas cerebrales, los impulsos eléctricos entran por las dendritas, sufren una transformación en el soma y se conectan a la siguiente neurona a través del axón, el cuál está conectado con la dendrita de la próxima neurona. Para las NN, las dendritas representan un vector de entrada, el soma representa una función que se le aplicará a dicho vector y el axón es análogo a la salida de la neurona artificial. Esta comparación analógica se muestra en la figura 2.8.

Las NN sirven para predecir el comportamiento de ciertas variables de salida en base al conocimiento de otras variables de entrada, donde la relación que tienen estas es desconocida. En otras palabras, estas redes tiene como objetivo generar una función que relaciona a estas variables. Para hacerlo, se requiere primero configurar la estructura de la red y luego entrenar a dicha red. Para entender todo lo relacionado a las NN utilizadas en este trabajo primero se introducirán ciertos conceptos necesarios para entender las NN, a continuación se describirán diferentes NN específicas y finalmente se explicará la forma en que pueden entrenarse,

incluyendo la descripción de algunos métodos de optimización y de regularización.

2.3.1. Elementos transversales a las NN

En esta sección se presentan conceptos y elementos que se utilizan en las NN.

- **Grafos Computacionales**

  Para entender más en profundidad las NN es necesario introducir los grafos computacionales, ya que estos ayudan a comprender las estructuras y el álgebra que constituyen a las NN.

  Un grafo computacional es, por un lado, un esquema gráfico de una red, y por otro, la representación de las relaciones matemáticas existentes entre sus partes. En un grafo computacional se tienen dos tipos de elementos: los **nodos** y las **flechas** que unen a los nodos. Los nodos representan una o más funciones que son aplicadas al número que entrega la flecha de entrada y entrega el valor de dicha función a la flecha de salida. Por otro lado, las flechas, además de mostrar el flujo de información, pueden representar aplicación de funciones a la información de entrada u operaciones entre diferentes flechas que entran en una misma neurona siguiente. Lo descrito se muestra gráficamente en la figura 2.9.

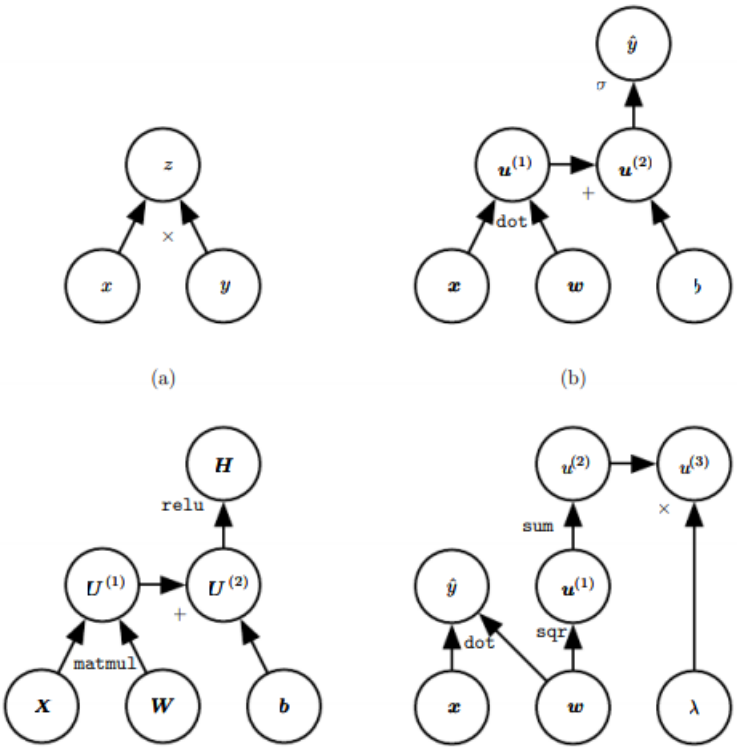

Figura 2.9: Ejemplos de grafos computacionales (adaptada de [51]).

Luego, los grafos computacionales son un elemento muy útil para visualizar el flujo de

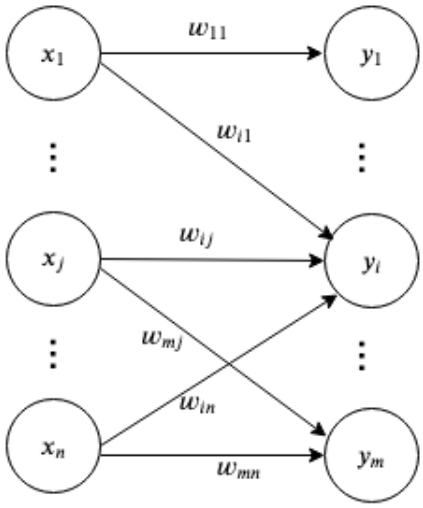

Figura 2.10: Grafo computacional de conexión entre dos capas de una red neuronal.

la información por la red y por ende comprender como dicha red está funcionando, lo cuál facilita el diseño y el análisis de ésta.

- **Función de Activación**

  Para comprender la función de activación es necesario definir que los **pesos** w son parámetros a ajustar por la red, que multiplican el valor de un nodo para entregar el resultado a la función de activación. Además, el **umbral** o **bias** b es otro parámetro a ajustar que se le entrega a cada función de activación para desincentivar o incentivar la activación de la neurona correspondiente.

  Luego, la función de activación tiene como entrada la suma del umbral y las multiplicaciones de los pesos por sus respectivos valores de activación de los nodos anteriores que están conectados con la neurona actual (ec. 2.16). La función de activación define si una neurona es o no activada, o en qué medida es activada.

$$a_{\mathrm{i}} = \sum_{j=1}^{n} w_{\mathrm{i}j} x_j - b_{\mathrm{i}} \tag{2.16}$$

  En esta ecuación el índice i corresponde a la neurona en la que se está aplicando la función de activación, mientras que el índice j corresponde a cada neurona de la capa anterior que está conectada con la neurona i, así, el peso $w_{\mathrm{i}j}$ es el que conecta a la neurona anterior j con la neurona i, que es de la que estamos obteniendo su activación. b_{i} es el umbral de la neurona actual. Esto puede verse en el grafo computacional de la figura 2.10.

De esta forma, la activación de la neurona actual estaría dada por (ec. 2.17):

$$y_{\mathrm{i}} = f(\sum_{j=1}^{n} w_{\mathrm{i}j} x_j - b_{\mathrm{i}}) \tag{2.17}$$

Y finalmente, la ecuación 2.17 escrita de manera matricial, representando así la capa completa de la red, se muestra en la ec. 2.18:

$$\vec{y} = f(\boldsymbol{W}\vec{x} - \vec{b}) \tag{2.18}$$

Dónde:
$\boldsymbol{W}$: Matriz de pesos.
$\vec{x}$: Vector de características de entrada.
$\vec{y}$: Vector de salida.
$\vec{b}$: Vector de umbrales.

Las funciones de activación más usadas son la función sigmoide (figura 2.11), la función tangente hiperbólica (figura 2.12) y la función ReLU, del inglés *Rectified Linear Unity* (figura 2.13). Estas se definen matemáticamente y con sus respectivas gráficas a continuación:

◇ **Función Sigmoide:**

$$f(x) = \frac{1}{(1 + \mathrm{e}^{-x})} \tag{2.19}$$

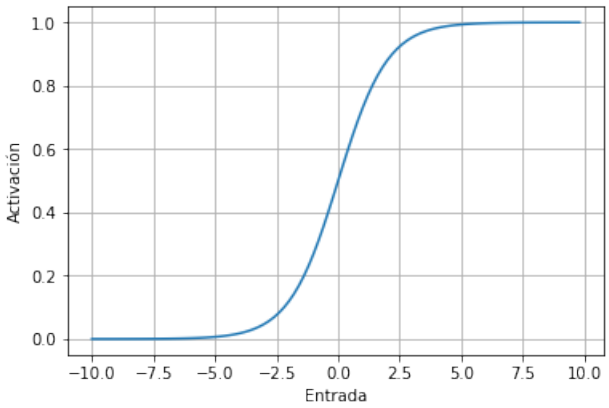

Figura 2.11: Función Sigmoide (ec. 2.19).

◇ **Función Tangente Hiperbólica:**

$$tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \tag{2.20}$$

◇ **Función ReLU**

$$f(x) = max(x, 0) \tag{2.21}$$

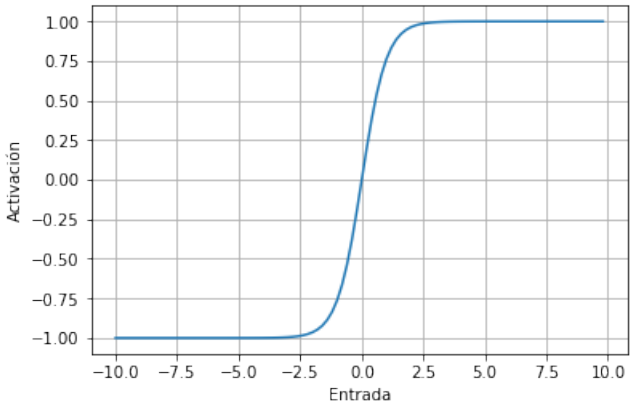

Figura 2.12: Función Tangente Hiperbólica (ec. 2.20).

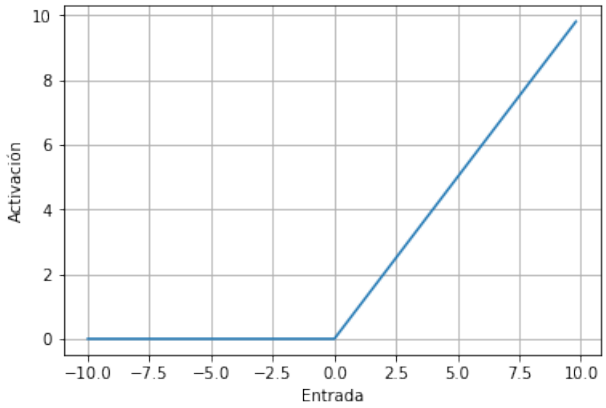

Figura 2.13: Función ReLU (ec. 2.21).

2.3.2. Red Neuronal Unidireccional (*NN feed-forward*)

A pesar de que en este trabajo no se utilizan las NN feed-forward, es necesario entenderlas antes de entender las otras redes más complejas que sí serán utilizadas.

Como se explicó, una red neuronal es una estructura que tiene una entrada y una salida. Esta estructura está compuesta por nodos que están conectados entre ellos. En una NN *feed-forward* o *Vanilla*, estos nodos están dispuestos en capas ordenadas secuencialmente, una capa tras otras. Los nodos de cada capa están conectados con los nodos de la capa siguiente. En la figura 2.14 puede apreciarse la estructura básica de una red neuronal *feed-forward*. Existen tres tipos de capas dependiendo de su posición secuencial. De izquierda a derecha, en la primera capa se copia la información del vector de entrada, de tal manera que a cada nodo se le asigna un valor de dicho vector. Esta capa se denomina **capa de entrada**.

A continuación están las **capas ocultas**, cuya cantidad y número de nodos que posee cada una son hiper-parámetros a definir al momento de configurar el modelo. La información fluye a través de estas capas activando las neuronas utilizando la función de activación de la capa en cuestión, al igual que en la última capa o **capa de salida**, donde se obtiene la respuesta del modelo. En cada nodo se aplica una función de activación a la entrada de dicho nodo.

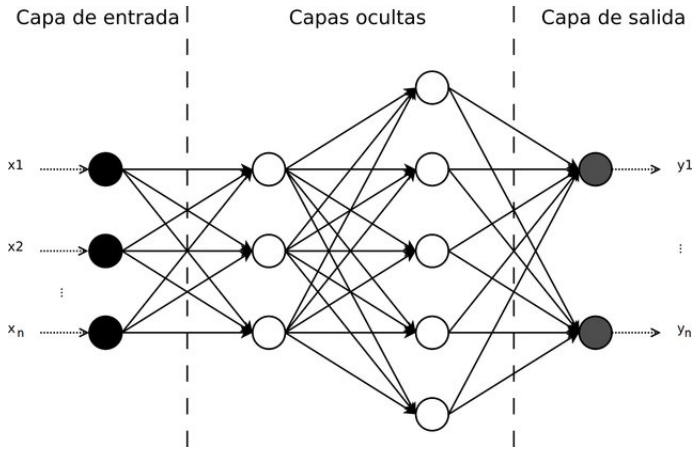

Figura 2.14: Estructura básica de una red neuronal (adaptada de [6]).

La cantidad de neuronas en la capa de entrada corresponde a la cantidad de características que tiene cada ejemplo mientras la cantidad de neuronas de la capa de salida, en el caso de una clasificación, corresponde a la cantidad de clases a las que puede corresponder la salida de cada muestra, mientras que en una regresión corresponde a la cantidad de valores de salida que se quiere predecir.

Matemáticamente, una NN *Vanilla* representa una composición de funciones aplicada al vector de entrada, siendo cada una de las funciones representadas por una de las capas ocultas. Tomando como ejemplo la red de la figura 2.14 y definiendo $g(\vec{x})$ como la función de activación aplicada al vector de entrada y $f(\vec{h})$ la función de activación aplicada al vector que arroja la primera capa oculta, entonces finalmente se tiene la ecuación 2.22, que representa a dicha NN:

$$\vec{y} = f(g(\vec{x})) \tag{2.22}$$

2.3.3. Red Neuronal Recurrente (RNN)

Una red neuronal recurrente es un tipo de NN especializada en procesar datos secuenciales ($[\vec{x}^{(1)}, \vec{x}^{(2)}, ..., \vec{x}^{(t)}, ...\vec{x}^{(L)}]$), ya que, además de tener una entrada y una salida en cada capa, tiene un **estado** ($\vec{h}$), el cuál realimenta a la misma capa en la siguiente iteración, generándose así memoria, a diferencia de la NN, donde cada ejemplo no tiene relación con el anterior. Esto puede verse en el esquema 2.15, donde a la izquierda está la forma conceptual de una red neuronal recurrente, mientras que a la derecha está la red desplegada que permite entenderla como una NN *Vanilla*, donde la propagación hacia atrás se realiza sobre una célula, pero a medida que se itera sobre la misma. En este caso estamos en presencia de propagación hacia atrás a través del tiempo (BPTT del inglés *back propagation trough time*).

Las ecuaciones que definen a una RNN Vanilla son la 2.23 y 2.24, dónde se puede ver que funciona como una NN Vanilla, pero además de tener pesos asociados a las entradas $\vec{x}$, representados por la matriz $\boldsymbol{U}$, tiene pesos asociados a los estados $\vec{h}$ representados por la matriz $\boldsymbol{W}$. También puede apreciarse que la función de activación usada es la tangente

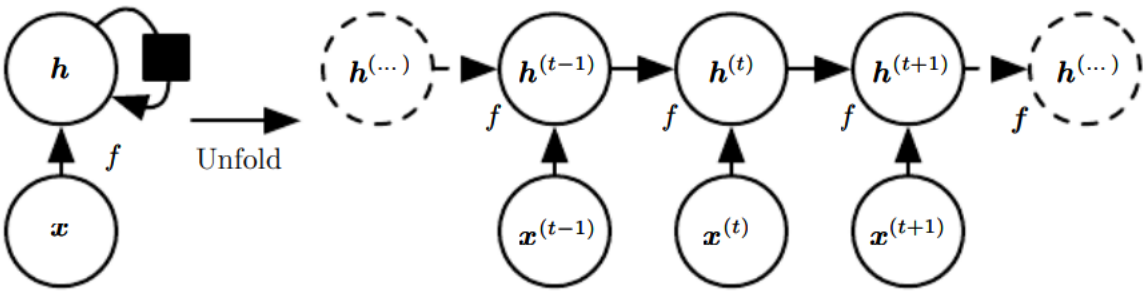

Figura 2.15: Diagrama de red neuronal recurrente (adaptada de [51]).

hiperbólica, aunque podría usarse otra.

$$\vec{y}^{(t)} = \vec{b} + \boldsymbol{W} \cdot \vec{h}^{(t-1)} + \boldsymbol{U} \cdot \vec{x}^{(t)} \tag{2.23}$$

$$\vec{h}^{(t)} = \tanh(\vec{y}^{(t)}) \tag{2.24}$$

En esta configuración, si se realizan muchas iteraciones, se cae en el **problema del desvanecimiento o explosión del gradiente descendente** [58, 14, 15], ya que se va multiplicando hacia atrás por los mismos parámetros y esto hace que el gradiente arroje resultados tendientes a cero o a infinito.

2.3.4. Red Neuronal Recurrente con Memoria de Largo y Corto Plazo (RNN-LSTM)

Las RNN-LSTM (del inglés *Recurrent Neural Network - Long-Short Term Memory*, al igual que las RNN, permiten secuencias de largo variable y poseen memoria, pero a diferencia de éstas superan el problema del desvanecimiento del gradiente descendente, ya que su dinámica selecciona las características que atraviesan las iteraciones y en que medida lo hacen, debido a que tienen sub-unidades encargadas de olvidar elementos de los vectores de información y además otras sub-unidades que seleccionan los elementos que salen del bloque LSTM completo.

Estas redes se basan en la repetición de un bloque que tiene diferentes compuertas. En la figura 2.16 puede verse una unidad de LSTM, la cual es un bloque complejo que tiene compuertas, dónde cada compuerta está asociada a una función de activación y está diseñada para cumplir cierta tarea, como se describe a continuación:

Input: Es la compuerta básica, que también se considera en una red NN o RNN *Vanilla*.

Input gate: Es la compuerta que modula la entrada, es decir, que dice cuánto del valor de entrada efectivamente entra en el bloque.

Forget gate: Es la compuerta que pondera el estado interno (s) del bloque.

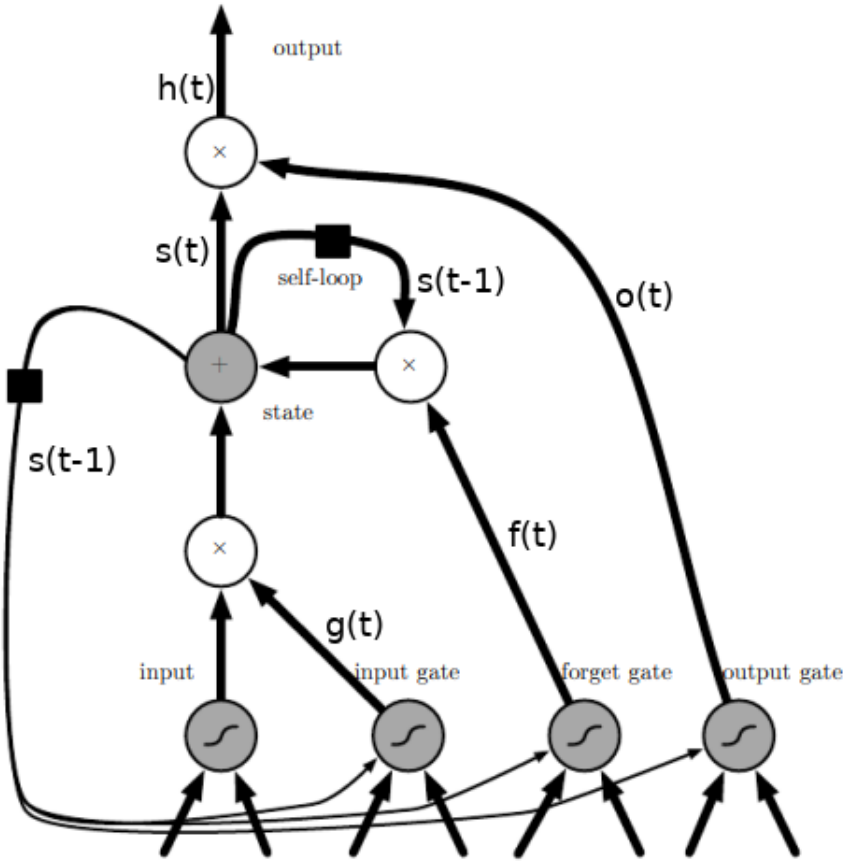

Figura 2.16: Esquema de RNN-LSTM (adaptada de [51]).

Output gate: Compuerta que pondera, en cada instante t, los elementos del estado interno (s), para obtener el estado (h).

Las ecuaciones de la salida de cada compuerta y de los estados se muestran a continuación:

$$f_{\mathrm{i}}^{(t)} = \sigma(b_{\mathrm{i}}^{f} + \sum_{j} U_{\mathrm{i},j}^{f} \cdot x_{j}^{(t)} + \sum_{j} W_{\mathrm{i},j}^{f} \cdot h_{j}^{(t-1)}) \tag{2.25}$$

$$s_{\mathrm{i}}^{(t)} = f_{\mathrm{i}}^{(t)} \cdot s_{\mathrm{i}}^{(t-1)} + g_{\mathrm{i}}^{(t)} \cdot \sigma(b_{\mathrm{i}} + \sum_{j} U_{\mathrm{i},j}^{s} \cdot x_{j}^{(t)} + \sum_{j} W_{\mathrm{i},j}^{s} \cdot h_{j}^{(t-1)}) \tag{2.26}$$

$$g_{\mathrm{i}}^{(t)} = \sigma(b_{\mathrm{i}}^{g} + \sum_{j} U_{\mathrm{i},j}^{g} \cdot x_{j}^{(t)} + \sum_{j} W_{\mathrm{i},j}^{g} \cdot h_{j}^{(t-1)}) \tag{2.27}$$

$$h_{\mathrm{i}}^{(t)} = \tanh(s_{\mathrm{i}}^{(t)}) \cdot q_{\mathrm{i}}^{(t)} \tag{2.28}$$

$$o_{\mathrm{i}}^{(t)} = \sigma(b_{\mathrm{i}}^{o} + \sum_{j} U_{\mathrm{i},j}^{o} \cdot x_{j}^{(t)} + \sum_{j} W_{\mathrm{i},j}^{o} \cdot h_{j}^{(t)}) \tag{2.29}$$

Donde cada compuerta tiene sus propios umbrales y pesos tanto para las entradas como para los estados, indicado por sus superíndices. Es importante diferenciar que hablamos de dos estados, h es el **estado principal**, que toda red neuronal recurrente tiene, la salida se conecta con la entrada directamente a través de h. Por otro lado tenemos el **estado interno** s, que realimenta a la célula sin considerar la compuerta output. Además, puede notarse en las ecuaciones que todas las funciones de activación son sigmoides, exceptuando la del estado principal, que es tangente hiperbólica. Una última aclaración es sobre las entradas. Como puede verse en la figura 2.16, todas las compuertas tienen dos entradas, estas entradas son las mismas para cada compuerta, por lo que es el bloque el que tiene dos entradas. Una de ellas es el estado principal, como ya fue mencionado, y la otra es el vector de características del conjunto de datos del problema. Además, puede verse que *input gate*, *forget gate* y *output gate* tienen una tercera entrada, el estado interno. Esta entrada es opcional en la definición de este algoritmo [51].

2.3.5. Autocodificador con eliminación de ruido (DAE - *De-noising Auto-Encoder*)

Es un método no supervisado de modificación de los datos de entrada que usa una o más capas de una red neuronal, su objetivo, además de retener parte importante de la información de entrada y estar restringido a la forma de la entrada, es maximizar la rubustez a la corrupción parcial de la entrada, es decir, entradas parcialmente corruptas debiesen arrojar casi la misma salida luego de la aplicación del Autocodificador. Un autocodificador entonces, consta de dos funciones, un codificador, que se aplica sobre el vector de entrada $\vec{x}$ y un decodificador, que se aplica sobre el vector codificado, el cuál se representa como la capa oculta por $\vec{h}$. El decodificador entrega la reconstrucción de la entrada (ver figura 2.17).

Específicamente en DAE, primero se realiza un mapeo estocástico de $\vec{x}$ (características de entrada) a $\tilde{\vec{x}}$ (mapeo de x). A cada una de las muestras de $\vec{x}$ se le aplica algún tipo de ruido,

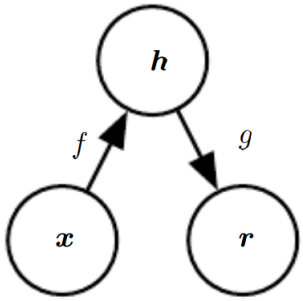

Figura 2.17: Autocodificador básico (adaptada de [51]).

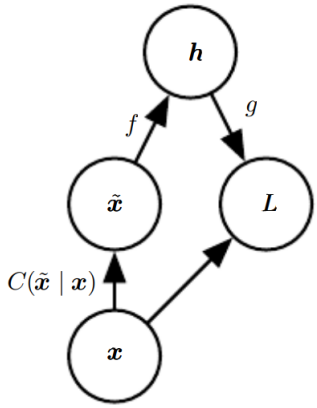

Figura 2.18: DAE (adaptada de [51]).

por ejemplo ruido gaussiano a todos los elementos del vector o la eliminación total de una proporción fija n de elementos en posiciones aleatorias del vector, obteniéndose así el vector corrupto de entrada ($\tilde{\vec{x}}$). Estos son los datos que ingresarán al algoritmo de codificación (ec. 2.30) y luego reconstrucción (ec. 2.31) y se itera para ajustar los parámetros (θ y θ') de este algoritmo para minimizar la pérdida entre la salida reconstruida y la data de entrada no corrupta, como se muestra en la ecuación 2.32. El grafo asociado a DAE se muestra en la figura 2.18.

$$\vec{h} = f_\theta(\tilde{\vec{x}}) \tag{2.30}$$

$$\vec{r} = g_{\theta'}(\vec{h}) \tag{2.31}$$

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} L(\vec{x}, g_{\theta'}(f_\theta(\tilde{\vec{x}})) \tag{2.32}$$

2.3.6. Optimización

En esta sección se presentan los conceptos necesarios para comprender como funciona la optimización en ML de manera general, además de diferentes métodos para realizarla.

- **Función de Pérdida**

  Todos los algoritmos de aprendizaje de máquinas se basan en maximizar o minimizar una función objetivo, esta se compone de la función de pérdida más un bias de regularización opcional.

  La función de pérdida (*Loss Function*) es la función objetivo a minimizar que dice qué tan errada fue la clasificación o regresión. Existen diferentes funciones de pérdida dependiendo de si se está haciendo una clasificación o una regresión. Algunas funciones de pérdida se muestran en la siguiente tabla (table 2.1):

Clasificación	**Regresión**
Log Loss	Mean Square Error / Quadratic Loss (L2 Loss)
Focal Loss	Mean Absolute Error (L1 Loss)
KL Divergence / Relative Entropy	Huber Loss / Smooth Mean Absolute Error
Exponential Loss	Log cosh Loss
Hinge Loss	Quantile Loss

Tabla 2.1: Diferentes funciones de pérdida para Clasificación y Regresión.

  En este trabajo se utiliza una regresión, por lo que se describirán las funciones de pérdida de ese tipo de red.

  - ◇ **L2 Loss:** El error cuadrático medio o pérdida cuadrática es el promedio de los errores cuadráticos en la salida, es decir:

$$Pérdida L2 = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.33}$$

    Dónde:

    N : Número de neuronas de salida.
    $\hat{y}_i$: Predicción de la salida que entrega la red para la neurona i de la capa de salida.
    y_i : Salida real obtenida de los datos de entrenamiento.
    Si esta medida tuviese un elemento representaría una parábola (ver figura 2.19) y esta tendría derivadas continuas. En el caso en que tiene varios elementos sus derivadas parciales son continuas, esto pone en ventaja a esta función de pérdida en comparación a L1 Loss, como se explica en 2.3.6.

  - ◇ **L1 Loss:** El error absoluto promedio es el promedio de la diferencia absoluta entre la salida predicha y la salida real, como se muestra a continuación en la ecuación 2.34:

$$Pérdida L1 = \frac{1}{N}\sum_{i=1}^{N} \mid \hat{y}_i - y_i \mid \tag{2.34}$$

    Su gráfica, considerando una sola neurona de salida, se muestra en la figura 2.20. Como se puede apreciar en la figura anterior, la pérdida L1 no tiene derivadas

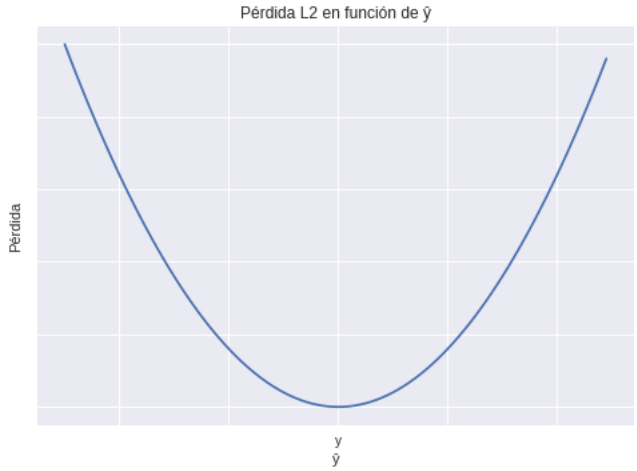

Figura 2.19: Función de pérdida L2.

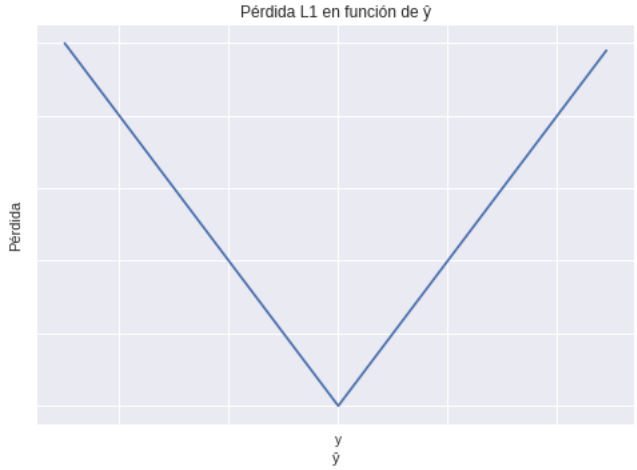

Figura 2.20: Función de pérdida L1.

continuas.
Esta función de pérdida es adecuada cuando hay valores atípicos (Outliers) que son datos corruptos, ya que cuando se le aplica gradiente descendente no aumenta tanto el cambio de los parámetros.

- **Gradiente Descendente**

  Es un método de cálculo que sirve para acercarse al valor mínimo de una función de manera iterativa, donde en cada paso los parámetros de la función se modifican hacia la dirección contraria del gradiente de dicha función, cómo puede verse gráficamente en la figura 2.21. La cantidad en que se modifican es la multiplicación del gradiente por la **tasa de aprendizaje**, que es el hiper-parámetro que pondera el cambio de los parámetros en cada iteración. Esto puede expresarse en la siguiente ecuación:

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta) \tag{2.35}$$

  Dónde:

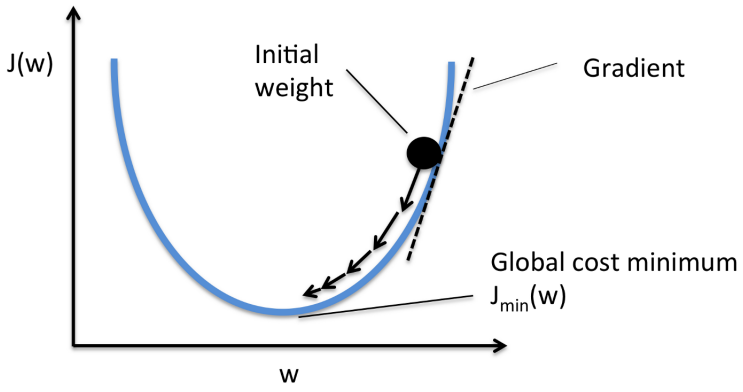

Figura 2.21: Funcionamiento de gradiente descendente simplificada, con un sólo parámetro a optimizar (w).

***θ* :** Vector de parámetros que son argumentos de la función J.
J: Función de costos a minimizar.
***η* :** Tasa de aprendizaje.

Existen variantes del gradiente descendente dependiendo de la cantidad de datos que se utilizarán para ejecutar el gradiente descendente. Algunos de ellos se explican a continuación.

- **Gradiente descendente por lotes (BGD -*Batch Gradient Descent*):** También conocido como gradiente descendente Vanilla, es la versión que computa el gradiente descendente de toda la muestra y los promedia para entregar el gradiente, por este motivo, si la cantidad de datos es muy grande, exige mucha memoria y el proceso es más lento, ya que se requiere calcular los gradientes de toda la data. Además, esta variante no permite actualizar el modelo online, es decir, agregar data después de que la red comenzó a entrenarse.
  El gradiente descendente por lotes garantiza la llegada al óptimo global en funciones convexas y al óptimo local en funciones no convexas [103].

- **Gradiente descendente estocástico (SGD - *Stochastic Gradient Descent*):** En contraste con BGD, esta variante ejecuta la actualización de parámetros en cada ejemplo de entrenamiento, por lo que suele ser más rápido ya que no tiene que calcular el gradiente para toda la data de una sola vez [102]. La ecuación 2.36 expresa su funcionamiento:

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.36}$$

  Dónde:
  $\boldsymbol{x^{(i)}}$ **:** Entrada del ejemplo i-ésimo de entrenamiento.
  $\boldsymbol{y^{(i)}}$ **:** Salida del ejemplo i-ésimo de entrenamiento.

- **Gradiente descendente por mini-lotes (*Mini-batches*):** Esta versión del gradiente descendente mezcla las dos anteriores, separa los datos en subconjuntos más pequeños y en cada subconjunto aplica gradiente descendente por lotes, con el resultado de cada mini-lote actualiza los parámetros iterativamente [103], como

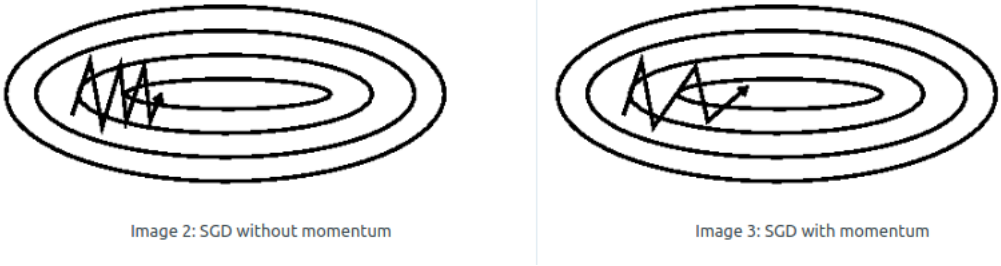

Figura 2.22: Mejora con Momento en Gradiente Descendente (adaptada de [49]).

en SGD.

- **Algoritmos de optimización para el Gradiente Descendente:**

  Existen diferentes formas de ir actualizando los parámetros utilizando los gradientes, a continuación se describen los necesarios para llegar, gradualmente, a comprender el algoritmo **Adam**, que es el que se usa en este trabajo.

  - ◇ **Momento:** Cuando se está cerca de un mínimo local, pero con respecto a un parámetro la pendiente es baja y con respecto a otro la pendiente es alta, entonces el SGD comienza a oscilar en torno al mínimo con respecto al parámetro que tiene alta pendiente y con respecto al otro va avanzando muy lento. El Momento soluciona este problema modificando el cambio de la iteración actual, sumándole una fracción (γ) del cambio que se generó en la iteración anterior, así disminuye los cambios oscilantes y aumenta los cambios que van iterativamente en la misma dirección (ver figura 2.22 y ecuaciones ecuaciones 2.37 y 2.38).

    .

$$v_t = \gamma \cdot v_{t-1} + (1 - \gamma) \cdot \nabla_\theta J(\theta) \tag{2.37}$$

$$\theta_{t+1} = \theta_t - \eta \cdot v_t \tag{2.38}$$

    Dónde:
    η: Tasa de aprendizaje.
    γ: Es el momento y toma valores entre cero y uno, usualmente 0.9.
    v_t: Es el cambio neto que al multiplicarlo por η se resta del parámetro.
    θ : Representa todos los parámetros de la red en cuestión.
    J : Función objetivo que se quiere minimizar.
    ∇_θ : Operador que representa el gradiente con respecto a θ de la función a la que se esté aplicando, en este caso J.

  - ◇ **AdaGrad:** *Adaptative Gradiente* o AdaGrad [40], es un algoritmo que discrimina entre los parámetros que más afectan a las características que existen más y menos frecuentemente en la muestra, actualizando en mayor cantidad los parámetros

relacionados con las características menos frecuentes.

Se define:

$$g_t = \nabla_\theta J(\theta_t) \tag{2.39}$$

$$G_t = G_{t-1} + g_t^2 \tag{2.40}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t \tag{2.41}$$

Dónde:
g_t: Gradiente con respecto a los parámetros en cada iteración.
G_t: Acumulación de gradientes g_t hasta instante t.
ε: Factor muy pequeño para asegurar que la expresión no esté dividida por cero (por ejemplo 10^{-7}).
η: Tasa de aprendizaje.

◇ **RMSprop:** *Root mean square prop* o RMSprop [111] utiliza otra forma para ponderar la tasa de aprendizaje, que es una mejora de AdaGrad, ya que además utiliza momento para su actualización. En vez de tomar la suma acumulativa de los cuadrados del gradiente, toma su media móvil.

$$g_t = \nabla_\theta J(\theta_t) \tag{2.42}$$

$$E[g^2]_t = \gamma \cdot E[g^2]_t + (1 - \gamma) \cdot g_t^2 \tag{2.43}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} \cdot g_t \tag{2.44}$$

Dónde, además de usar la misma nomenclatura que para AdaGrad, se tiene que:
$E[g^2]_t$: Media móvil de los cuadrados de los gradientes hasta iteración t.

◇ **Adam [67]:** Es un algoritmo de optimización de primer orden, es decir, utiliza sólo primeras derivadas de la función objetivo con respecto a los parámetros. Está basado en el gradiente descendente y sirve para funciones objetivo estocásticas. Su implementación es directa, es eficiente computacionalmente, tiene un bajo requerimiento de memoria, es invariante al reescalamiento diagonal de los gradientes, es decir, si los gradientes se multiplican por un factor c, la actualización de parámetros no se ve afectada por esto. Es adecuado para problemas con muchos datos y/o parámetros.
Este algoritmo se basa en la estimación de los momentos de primer y segundo orden de los gradientes, de ahí que su nombre deriva de: *adaptive moment estimation* (Adam). Adam es otro metodo que pondera la tasa de aprendizaje, al igual que AdaGrad y RMSprop. Además de utilizar el promedio de los cuadrados de los gradientes anteriores (v_t, momento de segundo orden), como RMSprop, Adam tambien utiliza el promedio de los gradientes anteriores, de manera similar al algoritmo momento (m_t, momento de primer orden). Mientras que el momento puede verse como una bola corriendo por una pendiente, Adam puede verse como una bola pesada con roce. A continuación se muestran las ecuaciones respectivas:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.45}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.46}$$

Dónde m_t y v_t son estimaciones del primer momento (la media) y del segundo momento (la varianza descentrada) de los gradientes respectivamente, por eso el nombre del método. Cómo m_t y v_t son inicializados como vectores de 0's, los autores de Adam observaron que tenían un sesgo hacia cero, especialmente durante las iteraciones iniciales y también cuando las tasas de decaimiento son pequeñas (es decir, β_1 y β_2 cercanos a 1). Para contrarrestar estos sesgos se corrigen los momentos según las siguientes ecuaciones:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.47}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.48}$$

Finalmente, usando los momentos no sesgados se actualizan los parámetros, como se muestra en la ecuación 2.49.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \cdot \hat{m}_t \tag{2.49}$$

Para una mayor claridad, a continuación se muestra el algoritmo:

El algoritmo requiere:
α: magnitud del paso
β_1 y $\beta_2 \in a[0, 1)$: Tasa de decaimiento exponencial para las estimaciones de los momentos.
$f(\theta)$: Función objetivo estocástica con parámetros θ
θ_0: vector de parámetros inicial.
Algoritmo:
$m_0 \leftarrow 0$(inicializa vector de 1^{er} momento)
$v_0 \leftarrow 0$(inicializa vector de 2^o momento)
$t \leftarrow 0$ (inicializa tiempo)
mientras θ_t no converge **hacer:**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$
fin bucle
retorna θ_t (parámetros resultantes)

Los autores recomiendan los siguientes parámetros:
$\beta_1 = 0{,}9$
$\beta_2 = 0{,}999$
$\varepsilon = 10^{-8}$

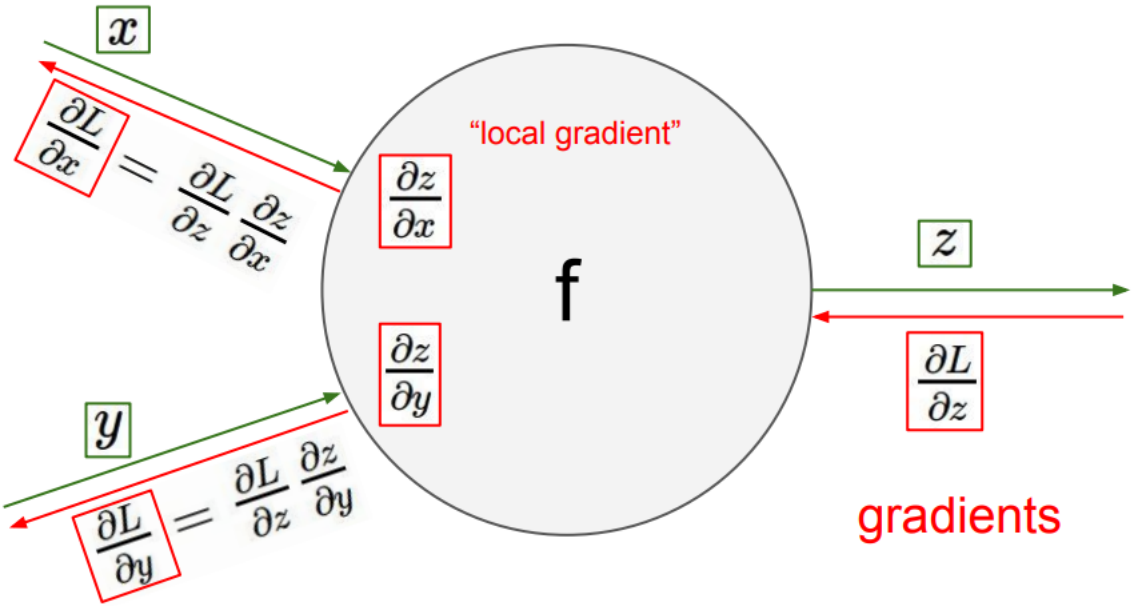

Figura 2.23: Flujo de propagación hacia atrás y hacia adelante en una neurona.

- **Propagación hacia atrás (*Back Propagation*):** El concepto de propagación hacia atrás viene de que en una red neuronal, primero, se realiza una propagación hacia adelante, es decir, el vector de entrada va atravesando la red hacia adelante hasta salir de la capa de salida y así obtener la predicción respectiva. Así mismo, la propagación hacia atrás parte del error que se obtuvo en esa predicción y recorre la red hacia atrás, hasta la capa de entrada. Para dejarlo más claro se presenta la figura 2.23, donde el flujo de propagación hacia adelante se muestra con flechas verdes y la propagación hacia atrás con flechas rojas. En dicha figura se muestra sólo una neurona de una capa, para simplificar el grafo.

  Cuando una red neuronal tiene varias capas de neuronas, para aplicar el gradiente descendente de la función de pérdida con respecto al peso de una neurona de una capa cualquiera, hay que utilizar la regla de la cadena en la derivación. Esto podría ser muy costoso si se obtuviera analíticamente la derivada de cada expresión y se calculara cada vez que fuese necesario, para eso sirve la propagación hacia atrás, ya que esta realiza la regla de la cadena optimizando los recursos computacionales, realizando derivaciones numéricas y guardando las expresiones que se repiten con sus respectivas derivadas.

  Para visualizar el proceso en su totalidad es importante notar que cada neurona se compone de dos operaciones, la primera es la suma de todos los pesos multiplicados por los valores de sus neuronas respectivas y a esto se le agrega el umbral b ($a = \vec{w}^T\vec{x}+b$), la segunda es la aplicación de una función no lineal a lo obtenido en la primera operación ($h = f(a)$). Estas dos operaciones serán derivadas en la regla de la cadena, para cada neurona que atraviesa la propagación hacia atrás, hasta la neurona que se le están modificando sus pesos de entrada. Como el resultado final de la red es la predicción y realizaremos la regla de la cadena con respecto a todos los pesos de la red y cada neurona depende de los vectores de entrada, que son valores dados en el entrenamiento, y los pesos, entonces la regla de la cadena tiene como variable solamente los parámetros de

la red. Esto puede verse a continuación, mostrando la derivada de la función de pérdida con respecto al peso:

$$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta y}\frac{\delta y}{\delta h}\frac{\delta h}{\delta w} \tag{2.50}$$

Considerando que la salida de cada neurona entra como input en varias neuronas de la capa siguiente, entonces se tendrá que sumar la aplicación de la regla de la cadena por los distintos caminos, esto se explicita en la ecuación 2.51 y viene a resumir la obtención de los gradientes usando *back propagation*, donde u^j son cada uno de los parámetros y u^n son cada una de las pérdidas en la salida.

$$\frac{\partial u^n}{\partial u^j} = \sum_R \prod_{k=2}^{t} \frac{\partial u^{(\pi_k)}}{\partial u^{(\pi_{k-1})}} \tag{2.51}$$

$R : recorrido(u^{\pi_1}, u^{\pi_2}, ..., u^{\pi_t}), \text{desde } \pi_1 = j \quad hasta\ \pi_t = n$

2.3.7. Regularización

En esta sección se mostrarán los métodos y estrategias que sirven para la regularización en ML, es decir, que sirven para disminuir el error de test, pero no el error de entrenamiento. De hecho, en la mayoría de los casos, al disminuir el error de test se incurre en un aumento del error de entrenamiento. Es importante recordar que un algoritmo de ML está diseñado para optimizar ciertos parámetros internos del modelo en cuestión. Además de estos parámetros hay parámetros que se refieren a la estructura y dinámica de entrenamiento, como es la cantidad de épocas que se le entregaran los datos al algoritmo para el entrenamiento, o las dimensiones de la red, es decir, la cantidad de capas y la cantidad de unidades en cada capa. Estas variables de configuración, y otras que modifican la forma en que aprende el algoritmo, se denominan **hiper-parámetros**.

Ya fue mencionado que los parámetros son entrenados utilizando el conjunto de entrenamiento y el desempeño de la red entrenada es testeada en el conjunto de test. Los hiper-parámetros no son entrenados por el algoritmo en sí, por lo que se requiere otra estrategia para hacerlo. Para esto, el conjunto de entrenamiento es separado en dos conjuntos, los que se denominan a su vez conjunto de entrenamiento y conjunto de validación. Puede surgir una confusión, ya que dos conjuntos se denominan con el mismo término (çonjunto de entrenamiento). Para aclarar esto se muestra el diagrama de la figura 2.24, dónde el conjunto más grande de entrenamiento es usado para ajustar los parámetros, mientras que el conjunto más pequeño, también denominado de entrenamiento (*training*), es utilizado para probar diferentes hiper-parámetros.

En resumen, se separan los datos en un conjunto de entrenamiento, otro de validación y otro de test. El conjunto de validación es utilizado para testear los diferentes hiperparámetros y seleccionar al que obtenga mejores errores en la validación y puesto que el conjunto de validación no fue utilizado para modificar los parámetros de la red, es que la elección de estos hiper-parámetros aporta a la generalización del algoritmo

- **Búsqueda de hiper-parámetros en cuadrícula (*grid search*):**

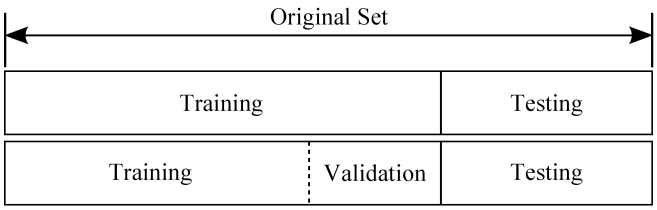

Figura 2.24: Visualización de la separación de los datos en los conjuntos de interés (adaptada de [20]).

Para encontrar los hiperparámetros que entreguen el error más bajo en la validación existen diferentes técnicas, donde una de las más simples es la búsqueda por cuadrícula.

En esta estrategia se define un conjunto de valores para cada hiper-parámetro y se busca en todas las combinaciones generadas por estos conjuntos. Por ejemplo, se quiere definir cuantas épocas entrenar una NN, cuántas capas debe tener la red y cuántas unidades debe tener cada capa, todo esto con la finalidad de aumentar la generalización. Entonces, decido iterar el entrenamiento usando 50 y 100 épocas, 2 y 4 capas; y 50 y 100 unidades por cada capa. En este ejemplo se tiene una cuadrícula de 3 dimensiones (épocas, capas y unidades por capa) y cada una de ellas podrá tener dos valores. Luego, habría que entrenar el algoritmo $2 \cdot 2 \cdot 2 = 8$ veces y escoger los hiperparámetros que entregaron un mejor error en la validación.

- **Enfoque multi-tarea (*multi-task approach*):**

Algunos problemas que puede resolver ML, se necesita predecir diferentes variables, donde dichas variables representan transformaciones diferentes de ciertas características de datos en común, por ejemplo si se quiere saber si una imagen contiene animales y por otro lado, si contiene electrodomésticos, sería útil representar los pixeles de la imagen de una forma común para ambas tareas, y luego hacer que el algoritmo se especialice en cada una. Esto puede verse en la figura 2.25, donde existen 3 tareas diferentes, pero las tres usan la misma representación generada por 3 capas iniciales de la red. Esto es el enfoque multi-tarea [23].

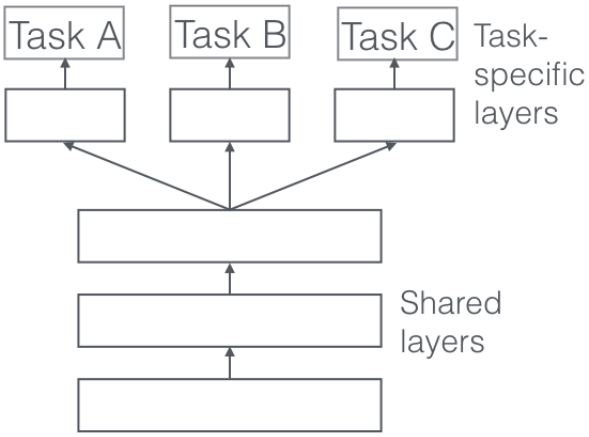

Figura 2.25: Diagrama enfoque multi-tarea (adaptada de [104]).

- **Aprendizaje semi-supervisado:** [25]

- **Detención Temprana (*Early-Stopping*):** Cuando se entrena un modelo con suficientes ejemplos como para sobreajustar los parámetros, entonces sirve la detención temprana, ya que esta evalúa en cada época el error en la validación y si este deja de disminuir durante una cantidad definida de épocas (**coeficiente de paciencia**), entonces el algoritmo guarda los parámetros con los que se obtuvo el mejor error de validación.

- **Mini-Lotes (*Mini-Batch*):**

  La agrupación de los ejemplos en mini-lotes puede aportar a la regularización, ya que de lo contrario los parámetros se actualizan con respecto a cada ejemplo, pudiendo sobreajustar con respecto a esos ejemplos

- **Aumento de los datos (*Data Augmentation*):**

  Una forma fácil de regularizar es simplemente aumentar la cantidad de los datos, introduciendo datos falsos, los que, en el caso ideal, debiesen ser generados por la distribución real de los datos, lo cuál es imposible en la mayoría de los casos, ya que no se conoce la distribución real, puesto que se generaron por procesos demasiado complejos. Peso a ello, es posible acercarse a datos que sean similares al conjunto de entrenamiento, por ejemplo sumando ruido gaussiano a los ejemplos del conjunto de datos original [115], dónde el problema está en escoger una media y varianza adecuada para el ruido. De esta forma el modelo tiene más datos, diferentes a los del conjunto inicial de entrenamiento, mejorando así la generalización. Además, hay pruebas de que las NN no son muy robustas al ruido [125], por lo que es adecuado entrenarlas usando ejemplos con ruido agregado.

  Está demostrado que esta estrategia es útil para reconocimiento de lenguaje hablado, por ende podría servir también para datos en formato de sonido [63].

- **Validación cruzada de k separaciones (*k-fold Cross-Validation*):**

  La validación cruzada es una estrategia que va cambiando el conjunto de validación en cada iteración, teniendo asi acceso a errores de validación producidos por diferentes subconjuntos de la muestra. esto puede verse en la figura 2.26.

  Esa estrategia sirve para seleccionar hiper-parámetros, evaluando el error de validación promedio utilizando diferentes hiperparámetros en el entrenamiento, para quedarse con la configuración que arroje un mejor resultado. Con los hiper-parámetros encontrados se procede a aplicar el algoritmo sobre el conjunto de test.

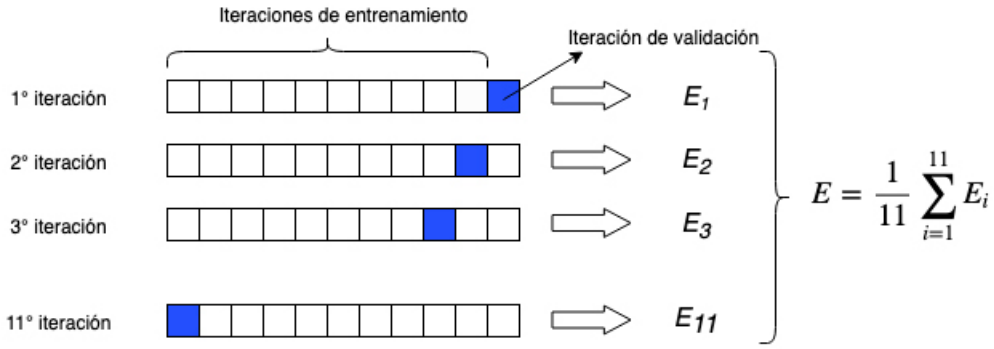

Figura 2.26: Validación cruzada de 11 separaciones.

Capítulo 3

Conceptos psicológicos inherentes a la base de datos DEAM

En esta sección se muestran conceptos no ingenieriles que serán utilizados en la solución del problema de ML.

3.1. Modelo Circunflejo de las Emociones (*Circumplex Model of Affect*):

El modelo Circunflejo de las emociones propuesto por Russel [1] se basa en la evaluación de dos variables continuas: **valencia** y **activación**, donde la primera se refiere a que tan agradado se siente el sujeto y la segunda se refiere a qué tan activo está. Un diagrama para entender estas variables se muestra en la figura 3.1. Es importante comprender esto, puesto que las variables de salida (o variables a predecir) del algoritmo ML son precisamente de este tipo.

Este modelo proporciona una herramienta útil que permite visualizar las emociones de un sujeto, lo que facilita su análisis.

3.2. Escala de Likert

La escala de Likert es una herramienta de medición que, a diferencia de preguntas dicotómicas con respuesta sí/no, nos permite medir actitudes y conocer el grado de conformidad del encuestado con cualquier afirmación que le propongamos. Resulta especialmente útil emplearla en situaciones en las que queremos que la persona matice su opinión. En este sentido, las categorías de respuesta nos servirán para capturar la intensidad de los sentimientos del encuestado hacia dicha afirmación.

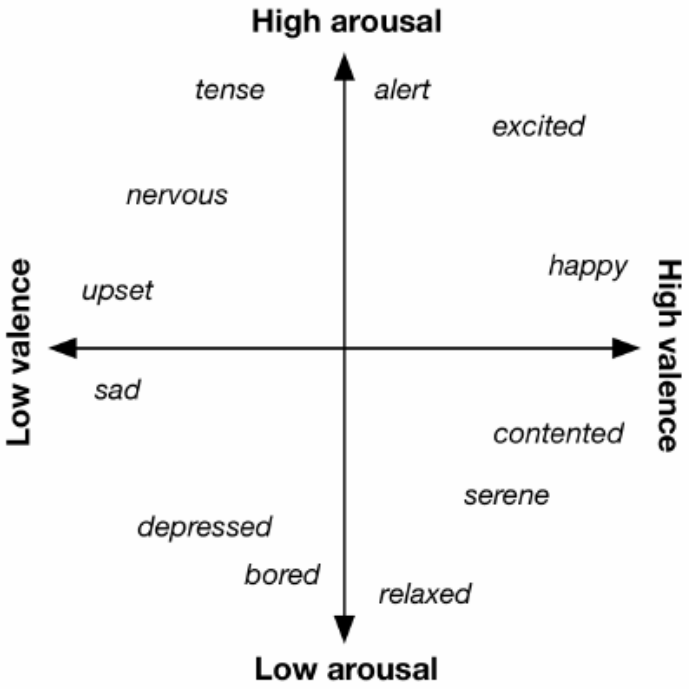

Figura 3.1: Diagrama Modelo Circunflejo de las Emociones (adaptada de [52]).

Sería un ejemplo de Likert la afirmación “Me agrada tomar cerveza caminando” y la escala de valoración: Totalmente de desacuerdo/ En desacuerdo/ Neutral/ De acuerdo/ Totalmente de acuerdo. En dicho ejemplo "Me agrada tomar cerveza caminandoçorresponde a un **ítem de Likert**.

La interfaz con que se recopilan las variables de salida (anotaciones), utiliza ítems de Likert (ver 6.1.1).

Capítulo 4

Teoría de la Información

En esta sección se presentan las herramientas concernientes a la caracterización que utiliza la teoría de la información para la obtención de la complejidad. Este capítulo está ordenado de manera orgánica, dando en primer lugar una mirada amplia del tema, para luego describir las herramientas que se utilizan efectivamente en este trabajo.

La teoría de la información, postulada por Shannon en 1948, se ha visto como una potencial herramienta para el análisis de la música y han existido diversos intentos en esta materia. Pese a ello, no se ha logrado elaborar una teoría potente que relacione la música y la Entropía, principalmente por que la música al parecer no es un fenómeno totalmente interpretable desde el punto de vista de las ciencias físicas, ya que su componente subjetiva es lo más importante y depende de la percepción de quién escucha. Por otro lado, en el desarrollo de una teoría musical con respecto a su información, han existido principalmente 3 desafíos que obstaculizan esta tarea: 1.- Es difuso el marco de preguntas que la Teoría de la Información (TI) puede responder al respecto, 2.- Existe una dificultad práctica para tabular las entidades musicales necesarias para el análisis desde el marco de la (TI), 3.- Existe incerteza con respecto al tipo de entidades musicales útiles para el análisis. Debido a esto se pretende analizar repertorios de música de cámara de diferentes compositores según ciertas medidas de la información, como la Entropía y algunos de sus derivados, aplicadas a diferentes características melódicas de la música.

Previo a [79], han existido varios esfuerzos por relacionar medidas de la información (medidas que se plantéan en TI) y la música. Nettelheim [87] realiza una revisión bibliográfica de publicaciones que contienen aplicaciones estadísticas útiles para la musicología, entre las cuales algunas utilizan la Teoría de la Informacíón. Los trabajos realizados por Pinkerton en 1956 [97], Meyer en 1957 [82], Youngblood en 1958 [135], Krahenbuehl y Coons en 1959 [70], Cohen en 1962 [28], Hiller y Fuller en 1967 [57], Knopoff y Hutchinson en 1981 [68] y en 1983 [69], Snyder en 1990 [119], Huron en 2006 [60] y Temperley en 2007 [128] representan los avances fundamentales en el análisis de la música utilizando medidas de la información.

Con respecto a la música, *."En la TI, la información se refiere a la libertad de elección que tiene un compositor en su trabajo son sus materiales o al grado de incertidumbre que un oyente siente en respuesta a los tonos que elige un compositor en el desarrollo de su composición"*

[135, p. 25]. Dado este entendimiento, se ha podido tomar la teoría de la información como una herramienta de composición [97], como un identificador de estilos de música [135, 68, 69] y como un mecanismo para comparar analíticamente secciones dentro de una pieza musical [57].

Existe un enfoque en Psicología y Ciencias del Aprendizaje denominado Aprendizaje Estadístico [106], este se refiera a la capacidad de extraer regularidades estadísticas del entorno, para así abstraer la estructura de su información. Ha sido estudiado que los niños logran dicha abstracción con respecto a ambientes visuales [46], lingüísticos [107] y musicales [108]. De esto puede concluirse que la Teoría de la Información podría modelar de alguna forma la percepción humana.

Wilhelm Wundt fue un psicólogo que planteaba una relación entre la intensidad de un estímulo y el posible placer que este pudiese provocar ([134]), diciendo que existe un óptimo de intensidad de estímulo, para el cual se generaría el mayor placer en quien percibe dicho estímulo. Este óptimo no está en los extremos, resultando en que estímulos muy intensos o muy poco intensos no resultan placenteros. La forma en esta curva no varía entre las personas, pero si puede desplazarse.

Berlyne [16] adaptó esta teoría diciendo que la variable que afectaría en la percepción sería el potencial de excitación (ver figura 4.1), definido como: *"la fuerza psicológica de un patrón de estímulos o grado al cual este puede perturbar y alertar al organismo"*... En la que estarían implicadas ...*"propiedades como la novedad, la sorpresa y la complejidad"* [17]. En definitiva, Berlyne afirma que las personas prefieren estímulos de moderada novedad y complejidad. Según Fred Lerdahl [73], la mejor música utiliza completamente el potencial de

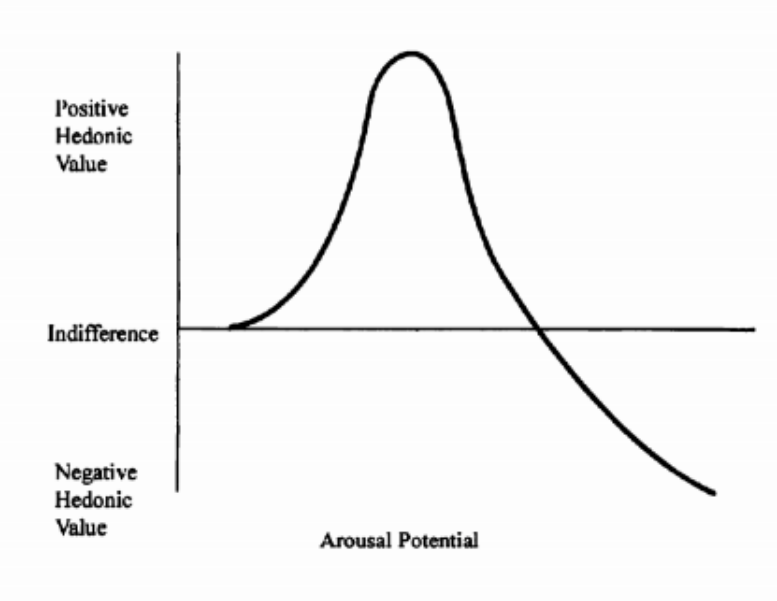

Figura 4.1: Curva de Wundt (adaptada de [16])

nuestros recursos cognitivos, y por otro lado, la mejor música surge cuando la gramática de la composición y la gramática de quién escucha coinciden.

En síntesis, la Entropía en la música, puede interpretarse como un buen identificador de

la potencialidad de excitación en los que escuchan[79]. A pesar que esto tiene mucho que ver con el sujeto que escucha, puede deducirse que una música de mayor complejidad, pero entendible por el sujeto, se puede categorizar como una "buena música". En otras palabras, si la música es demasiado compleja o demasiado simple para un oyente, esta no logra excitarlo positivamente.

Por lo anterior, es de interés la Entropía que tiene la música, para saber cómo nos puede afectar y así poder comprenderla según el punto de vista de su potencialidad de excitación.

4.1. Entropía de primer orden

La Entropía es el promedio de la cantidad de información necesaria para definir un evento aleatorio. Por lo tanto se refiere a qué tan impredecible es un sistema. La ecuación que entrega la Entropía es la siguiente (4.1):

$$H(X) = -\sum_{\mathrm{i}} p(x_{\mathrm{i}}) \log_2 p(x_{\mathrm{i}}) \tag{4.1}$$

Donde $p(x_{\mathrm{i}})$ es la probabilidad de que el elemento x_{i}, perteneciente al dominio de la variable aleatoria X, aparezca en el mensaje (evento aleatorio).

En el contexto de este trabajo, a la entropía se denomina entropía de primer orden. En este punto es necesario definir otras medidas de la información.

4.2. Entropía Conjunta y Entropía Condicional

Una vez definida la Entropía de una variable aleatoria, se extiende la definición a un par de variables aleatorias. No hay nada nuevo en esta nueva definición puesto que (X,Y) puede ser considerado como un único vector de variable aleatoria.

Definición: La Entropía Conjunta H(X,Y) de un par de variables aleatorias discretas (X,Y) con una distribución conjunta p(x,y) es definida como:

$$H(X,Y) = -\sum_{x \varepsilon X} \sum_{y \varepsilon X} p(x,y) \log_2 p(x,y) \tag{4.2}$$

Donde $\boldsymbol{y}$ corresponde a la nueva variable aleatoria perteneciente al dominio de $\boldsymbol{Y}$
A continuación se define la Entropía Condicional de una variable aleatoria dada otra, como el valor esperado de las Entropías de las distribuciones condicionales, en relación a la variable aleatoria condicionante.

Definición: Si (X, Y) $\sim$ p(x, y), la Entropía Condicional H (Y|X) es definida como:

$$H(Y|X) = -\sum_{x \varepsilon X} p(x) H(Y|X = x) \tag{4.3}$$

$$= -\sum_{x\varepsilon X} p(x) \sum_{y\varepsilon Y} p(y|x) \log_2 p(y|x) \tag{4.4}$$

$$= -\sum_{x\varepsilon X} \sum_{y\varepsilon X} p(x,y) \log_2 p(y|x) \tag{4.5}$$

Reescribiendo la ecuación anterior (ec. 4.5), en función del i-ésimo evento del mensaje, se obtiene la ec. 4.6.

$$H(X|Y) = -\sum_{\mathrm{i},j} p(x_\mathrm{i}, y_j) \log_2 p(x_\mathrm{i}, y_j) \tag{4.6}$$

Como puede verse en la ecuación 4.6, la Entropía condicional es el promedio de la cantidad de información necesaria para identificar cada evento x dado un evento y, y ponderando cada elemento de información por la probabilidad conjunta de dichos eventos.

$$H_{norm} = \frac{H_{actual}}{H_{máxima}} \tag{4.7}$$

Capítulo 5

Herramientas computacionales

En esta sección se describen superficialmente las herramientas computacionales, referentes a *hardware* y *software*, utilizadas en este trabajo.

5.1. GPU-CPU

En cualquier computador la Unidad Central de Procesamiento (CPU - *Central Processing Unit*) es la encargada principal del procesamiento de los datos necesarios para el funcionamiento del computador y sus aplicaciones. La Unidad de Procesamiento Gráfico (GPU - *Graphic Processing Unit* es la que fue diseñada inicialmente para procesar gráficos. Esta unidad es capaz de procesar muchos datos en paralelo, aplicando funciones específicas, relacionadas a lo gráfico, es decir a elementos matriciales y vectoriales. Por otro lado, la CPU está diseñada para efectuar operaciones más complejas que la GPU, pero en serie. Esta diferencia es debida a que una GPU tiene muchas unidades aritmética-lógicas (ALU's - *Arithmetic Logic Unit*), en cambio la CPU tiene sólo unas pocas, definida por el número de núcleos (*cores*) que tenga el procesador. Esta diferencia se muestra en la imagen de la figura 5.1.

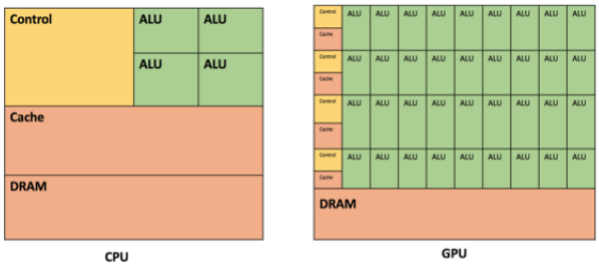

Figura 5.1: Comparación simplificada de arquitecturas CPU y GPU (adaptada de [9]).

En síntesis, la GPU es una pieza de *hardware* que sirve para acelerar el procesamiento de un computador, cuando se necesitan hacer muchas operaciones matemáticas en paralelo. Esto es idóneo para ML, ya que en esta disciplina muchas veces se requiere trabajar con matrices y vectores que tienen muchos datos y se les desea aplicar cierta sencilla operación, como se ha

mostrado en diferentes apartados de 2. Para poder usar esta capacidad de procesamiento, la empresa *Nvidia* ha desarrollado, como complemento a las GPU, una plataforma de desarrollo que sirve para poder programar cómo se utilizará dicha GPU, es decir, un *driver* programable que permite hacer que códigos escritos por el usuario sean ejecutados adecuadamente en la GPU. Esta plataforma se llama **CUDA** (*Compute Unified Device Architecture*). Lo que hace CUDA entonces, es distribuir las tareas que se definen por el usuario en los diferentes nodos de la GPU como puede verse en la figura 5.2

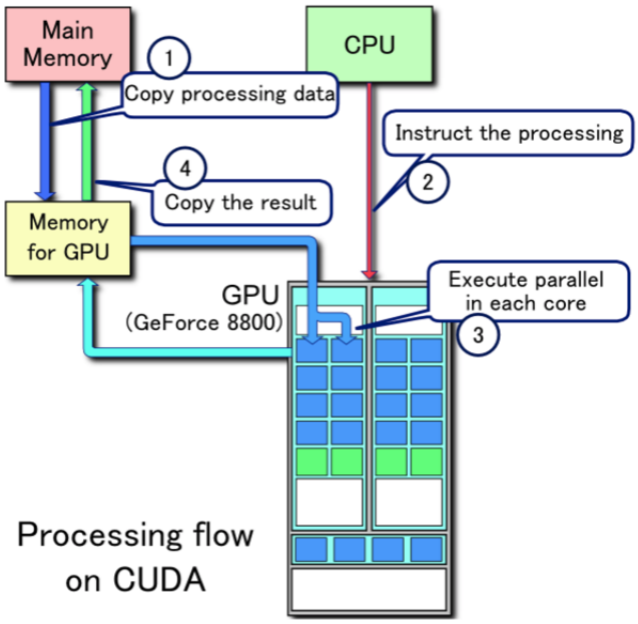

Figura 5.2: Flujo de trabajo definido con CUDA (adaptada de [133]).

Ciertas librerías de *Python* que sirven para ML utilizan CUDA para distribuir las operaciones de los algoritmos usados en diferentes celdas de la GPU. Como fue mencionado, para muchos casos de ML, usar la GPU puede mejorar el desempeño del algoritmo, pero esto depende de la estructura particular del algoritmo y de las especificaciones del *hardware* utilizado.

5.2. *Python*

Es el lenguaje de programación interpretado más usado en ML. Su sintaxis genera un código fácilmente legible y además es de código abierto, por lo que se ha popularizado entre desarrolladores y se han generado muchas librerías útiles no sólo para ML, si no que para la ciencia de los datos en general.

5.2.1. *numpy*

Librería de *Python* fundamental para las ciencias de la computación. Sirve para manejar datos numéricos de todo tipo. Contiene funciones matemáticas tanto simples como complejas, permitiendo trabajar adecuadamente con Álgebra Lineal.

5.2.2. *pandas*

Librería utilizada para analizar y visualizar bases de datos. Ampliamente utilizada en ML para la exploración y el filtrado de los datos en conjunto con *numpy*

5.2.3. *sqlite3*

Librería de *Python* que sirve como interfaz para utilizar SQL como base de datos. SQL tiene mejor desempeño que pandas para bases de datos grandes, por lo que en muchos casos es mejor utilizar sqlite3 en vez de pandas en ML, sobretodo en las partes del algoritmo dónde se guarda y obtiene la base de datos completa a utilizar.

5.2.4. *matplotlib*

Librería fundamental para graficar resultados en *Python*. Esta imita la sintaxis de *Matlab*. Existen varias librerías que se basan en *matplotlib*, por lo que de todas maneras es escencial para mostrar los resultados obtenidos en la ciencia de los datos.

5.2.5. *seaborn*

Librería para graficar, basada en *matplotlib* y que facilita la confección de gráficos más complejos y composiciones de gráficos de *matplotlib*.

5.2.6. *sklearn*

Scikit-learn es una librería construida sobre *numpy*, *scipy* y *matplotlib*. Diseñada para la minería de datos, el análisis de datos y ML. Tiene implementados algunos algoritmos de ML y herramientas útiles para analizar sus resultados.

5.2.7. *torch*

Pytorch es una librería para ML basada en torch. Tiene la particularidad de que, a diferencia de otras librerías de ML, como *TensorFlow* o *scklearn*, permite definir la arquitectura de la red durante su ejecución, siendo más flexible para cuando se está probando un algoritmo y/o la forma de los datos no se conoce a ciencia cierta.

5.3. *Colaboratory*

Colaboratory es un entorno gratuito de *JupyterNotebook* que no requiere configuración y se ejecuta completamente en la nube. Es posible acoplarlo a *Google Drive*, pudiendo entregar códigos y datos desde la nube personal (de *Google Drive*)para ser utilizados en la nube de *Colaboratory*. Es posible usar esta aplicación durante 12 horas de manera continua.

5.4. *Humdrum*

Humdrum es un *software* de propósito general que ayuda a investigadores de la música. *Humdrum* tiene muchas aplicaciones posibles, dónde las importantes para este trabajo son el análisis y transformación de archivos *.krn*, que son una forma simple de representar partituras musicales en un archivo de texto plano. De esta forma con *Humdrum* es posible obtener, por ejemplo, diferentes medidas de la información extrayéndolas de partituras.

5.5. Características específicas de *hardware*

A continuación se muestran las características de los computadores utilizados:

- ***Colaboratory:***

  GPU: 1 x Tesla K80 , 2496 núcleos CUDA, 12GB GDDR5 VRAM
  CPU: 1 x Un núcleo hyper threaded Xeon Processors @2.3Ghz i.e(1 núcleo, 2 hilos)
  RAM: 12.6 GB Disponibles
  Disco: 33 GB Disponibles

- ***PC personal***

  GPU: 1 x GeForce GTX 1060, 1280 núcleos CUDA, 6GB
  CPU: 1 x Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
  RAM: 16 GB Disponibles
  Disco: 75 GB Disponibles

Capítulo 6

Metodología

Este capítulo incluye la descripción de cómo se predijeron las características de la música, argumentando adecuadamente la elección de cada parte involucrada. Primero se detalla todo lo relativo al aprendizaje de máquinas utilizado en la caracterización de las emociones y luego lo utilizado en la obtención de las medidas entrópicas. La organización de cada sección es cronológica, describiendo paso a paso las decisiones involucradas, esclareciendo así el orden en que se tomaron dichas decisiones y al mismo tiempo el flujo de la información.

6.1. Predicción de emociones

Para obtener un modelo de regresión adecuado es necesario seguir una serie de pasos que son transversales en el aprendizaje de máquinas. En primer lugar hay que obtener los datos con que se entrenará la red. Luego es necesario explorar estos datos con lo cual se podría definir qué tipo de red sería adecuada. Con esta información se procede a preprocesar los datos y definir la forma en que se les entregará a la red. Luego es posible comenzar a entrenar nuestra máquina, lo que incluye la elección de hiperparámetros, la aplicación de métodos de regularización, y el entrenamiento en sí. Dicho entrenamiento consta de tres fases, una de *entrenamiento*, luego una de *validación*, las que se repiten iterativamente, para finalmente testear el modelo en una etapa de *prueba*, como fue explicado en 2.3.7. Cada una de estos pasos se describen y argumentan a continuación.

6.1.1. Descripción de la base de Datos DEAM (*Dataset for Emotional Analysis of Music*)

Base de datos compuesta principalmente por tres tipos de información: **1.** Música en formato *mp3*, **2.** Parametrizaciones de características físicas de la onda sonora de cada archivo *mp3* y **3.** Anotaciones de la emociones que induce la música de los archivos *mp3*.

Esta base de datos fue generada por *MediaEval* a partir de música *royalty-free* (licencia

Creative Commons que permite redistribuir el contenido) obtenida de *freemusicarchive.org* (FMA), *jamendo.com* y la base de datos *medleyDB*, seleccionada cuidadosamente con el objetivo de lograr un espectro variado desde el punto de vista de los estilos de la música y sus autores. Los estilos existentes en la base de datos son: *rock, pop, soul, blues, electrónica, clásica, hip-hop, internacional, experimental, folclórica, jazz, country, reggae* y *rap*. La base de datos está compuesta por 1802 archivos *mp3*: 1744 extractos de canciones de 45 segundos y 58 canciones completas. Es importante mencionar un detalle fundamental, los extractos de 45 segundos no comienzan cuando comienza la canción, si no que son un pedazo de una canción en algún lugar aleatorio de esta. Por otro lado, las canciones completas comienzan en el instante que comienza la canción.

La base de datos incluye diferentes parametrizaciones de **características** físicas de la música obtenidas con *OpenSmile*, *software open-source* que sirve para obtener características del sonido, como por ejemplo la frecuencia fundamental en un lapso de tiempo o el promedio de las amplitudes de la onda. Las características fueron extraídas en ventanas de tiempo de 0.5 segundos y ascienden a un total de 260 características en cada intervalo.

DEAM incluye **anotaciones** de la emoción que evoca la música según el modelo circunflejo de las emociones (3.1). Para obtener las anotaciones se utilizó la plataforma de *Amazon Mturk*, donde anotadores responden en línea a cuál es la **activación** y **valencia** que les produce una canción en cada instante (**anotaciones dinámicas**). En la figura 6.1 se muestra lo obtenido de un evaluador para la activación dinámica en *Amazon Mturk*. Además se le pregunta al anotador cuál es la activación y valencia de la canción en general (**anotaciones estáticas**). La interfaz donde responden se muestra en la figura 6.2. Estas anotaciones fueron generadas los años 2013, 2014 y 2015. Fueron 10 anotadores por cada uno de los 1744 extractos de 45 segundos el año 2013 y 2014. El año 2015 se seleccionaron 3 de ellos y 2 expertos de los laboratorios de *MediaEval* para generar las anotaciones de las 58 canciones completas. Es importante mencionar que las anotaciones de las canciones completas, existente en la base de datos, comienzan desde el segundo 15. De esta forma, las emociones de todas las canciones completas tienen una característica en común: comienzan 15 segundos después de que comienza la canción, a pesar de que los anotadores anotaron sus emociones desde el inicio de la canción; y por otro lado, en el caso de los extractos, los anotadores comenzaron a escuchar la canción en el mismo instante donde las anotaciones son agregadas en la base de datos. Esta diferencia puede traer una complicación al momento de predecir las emociones en la música, puesto que la forma en que se generaron fue distinta según lo mencionado.

En resumen, se tiene una base de datos que contiene extractos de canciones con sus respectivas características y anotaciones dinámicas, de valencia y activación cada 0.5 segundos, y sus anotaciones estáticas. La base de datos esta distribuida en diferentes archivos *mp3* y *csv*. El proceso de generación de los datos se muestra en la figura 6.3

6.1.2. Exploración de los Datos

Primero se visualizó la forma en que se entregan los datos. Para el caso de las características, cada uno de los archivos en *csv* corresponde a una canción y en ellos se encuentran tabuladas 260 características para cada intervalo de tiempo. En la tabla 6.1 se muestra el

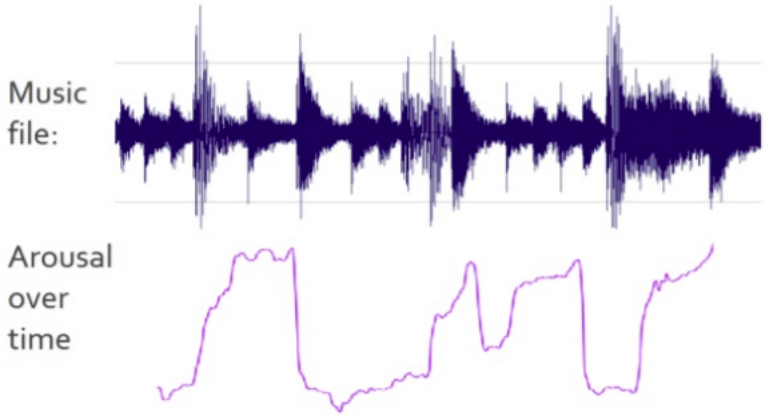

Figura 6.1: Ejemplo de datos obtenidos para la activación dinámica con la respectiva onda sonora (adaptada de [4]).

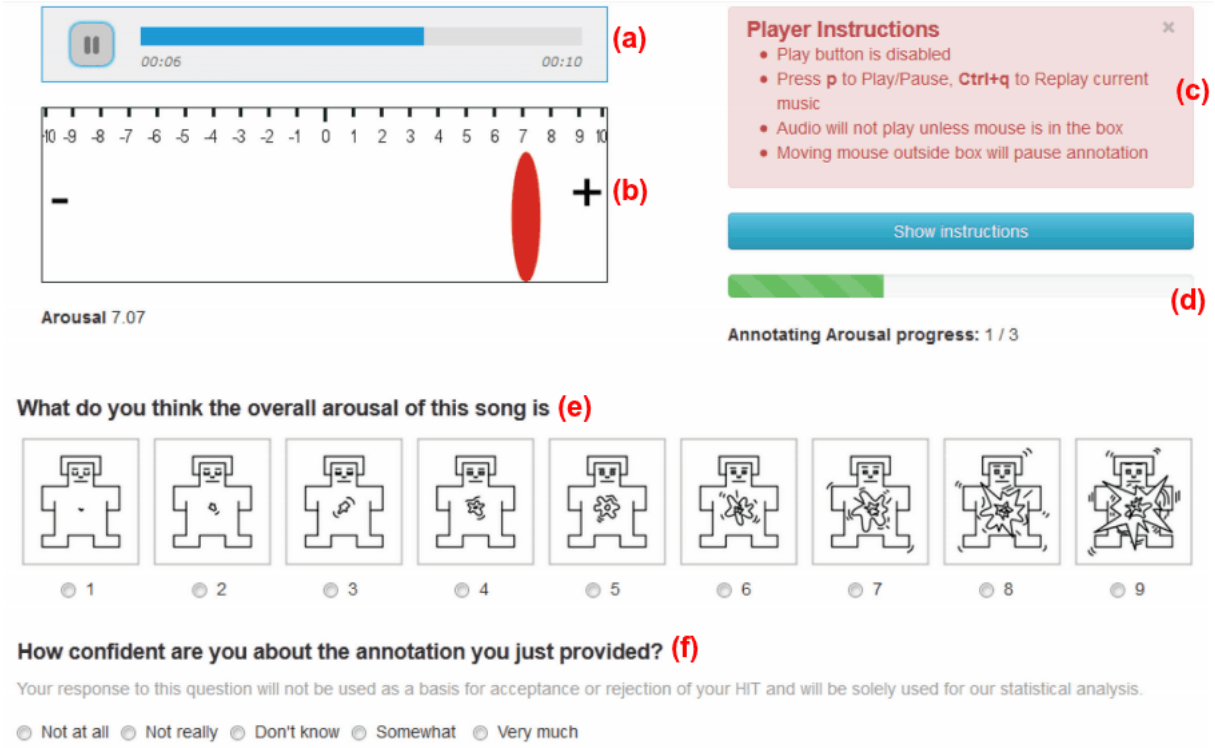

Figura 6.2: Interfaz *Amazon MTurk* (adaptada de [5]). Los usuarios la utilizan para generar las anotaciones (activación en este caso). (a): Control de reproducción. (b): Selector de nivel de emoción. En esta parte el usuario mueve el indicador rojo, utilizando el cursor, definiendo que emoción le provoca en cada instante (anotación dinámica). (c): Instrucciones. (d): Indica en que parte de la encuesta está (aún no responde la parte (e) ni la (f). (e): Ítem de *Likert* (3.2)con respecto a qué tanto excitó al sujeto la canción en su totalidad (anotación estática). (f): Aquí el sujeto explicita que tan seguro se siente con respecto a sus respuestas.

inicio de un archivo con las primeras 3 características en él, con el fin de familiarizar el lector

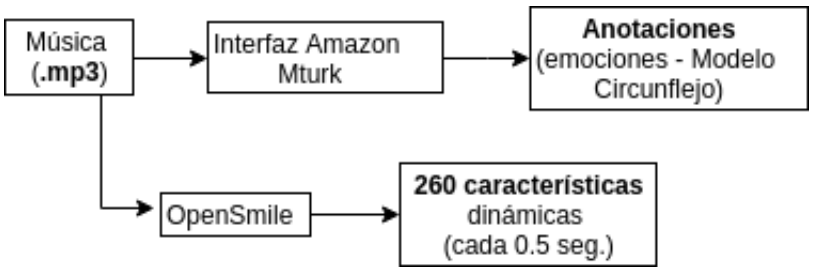

Figura 6.3: Proceso de generación de datos.

	frameTime	F0final_sma_stddev	F0final_sma_amean	voicingFinalUnclipped_sma_stddev
0	0.0	0.00000	0.00000	0.000000
1	0.5	46.61850	41.97005	0.378904
2	1.0	26.41163	83.31398	0.132493
3	1.5	20.73190	94.65977	0.020989
4	2.0	11.78187	104.48090	0.017434

Tabla 6.1: Características de la onda sonora (DEAM): Estas características fueron generadas con *OpenSmile*. La primera columna contiene el índice de cada fila. La segunda columna el inicio de cada intervalo en segundos. Las columnas siguientes corresponden a las primeras 3 características del archivo.

con la forma de los datos.

En dicha tabla puede verse, a priori, que las características están expresadas en diferentes escalas, para corroborar este hecho se generaron gráficos que muestran los rangos de dichas características y se obtuvieron las medias y desviaciones estándar de la media y la desviación estándar de las anotaciones estáticas, para mostrar objetivamente si era necesario normalizar los datos.

Para el caso de las emociones, primero se verificó si existía una relación entre valencia y activación estática de las canciones. En particular se graficaron por separado las relaciones para los 1744 extractos destinados al entrenamiento y para las 58 canciones destinadas al test. Además se obtuvieron las respectivas correlaciones de Pearson entre valencia y activación estática.

6.1.3. Pre-procesamiento de los Datos

Primero se observaron los datos para ver si existían anomalías y filtrarlas de algún modo adecuado. Los datos utilizados son las características físicas de la onda y las anotaciones dinámicas.

De las 1802 canciones y extractos de canciones se excluyeron las 58 canciones completas para utilizarlas como conjunto de test. Las 1744 restantes fueron filtradas basándose en el siguiente algoritmo, descrito en [3]:

1. Se obtuvo la correlación de Pearson entre las anotaciones de cada evaluador con respecto al promedio, y para los casos en que el resultado fue menor a 0.1, se eliminó la anotación. Si

una canción quedó con menos de 5 anotaciones, se eliminó de la base de datos.

2. Luego se descartaron las canciones que tenían un alfa de Cronbach menor que 0.6 entre las respuestas de los evaluadores que quedaron del paso **1.**.

3. En cada canción, se promediaron las anotaciones que quedaron del paso anterior.

4. Se modificó el promedio de todas las anotaciones de cada canción para que fuese igual al promedio de las anotaciones estáticas.

En la aplicación de este algoritmo surgieron diferentes decisiones adicionales, debido a la inexactitud en la explicación del preprocesamiento [3]. Primero: ¿era necesario borrar las anotaciones que contenían solo ceros?. Segundo: ¿Era necesario borrar las anotaciones en que r de Pearson resultaba indefinida?. Tercero: ¿Había que determinar el alfa de Cronbach a partir de las varianzas o a partir de las correlaciones?. Cuarto: ¿Había que borrar las canciones con alfa de Cronbach indefinido?. Y quinto: ¿Había que borrar las canciones con alfa de Cronbach negativo?

Se realizó el procesamiento para cada una de las combinaciones de decisiones referentes a las preguntas expuestas en el párrafo anterior, para evaluar qué conjuntos de datos se usarían en el resto del proceso.

Previo a esto fue necesario eliminar a ciertos anotadores, ya que no existía la misma cantidad en valencia y activación en algunas canciones.

6.1.4. Máquina de Vectores de Soporte (SVM)

Para ahondar en la comprensión de los datos y generar una primera aproximación a la clasificación de éstos, se aplicó SVC tomando como vector de entrada los promedios temporales de las características físicas de la onda obtenidos con OpenSmile y las respectivas clases fueron definidas como los 4 cuadrantes que se generan al dividir su valencia y activación estática según el modelo circunflejo, pero en vez de dividir cada variable en 5, que es el punto medio del rango de las variables; como se muestra en la figura 6.2 (b), se dividió donde está ubicado su promedio (ver figura 6.4). Además, para observar como se comporta SVC, se probó eliminando las canciones que tenían una valencia o activación muy cercana al promedio, según la ecuación 6.1.

$$Y = \left\{ y_{\text{i}} : y_{\text{i}} \geq \bar{Y} + 0{,}2 \ \wedge \ y_{\text{i}} \leq \bar{Y} - 0{,}2 \right\} \tag{6.1}$$

Se realizaron dichas pruebas con diferentes valores de γ y de C utilizando búsqueda por cuadrícula. Se utilizó como conjunto de entrenamiento y validación los 1744 extractos de canciones, aplicando validación cruzada de 5 separaciones y luego se testeó en las 58 canciones completas. La cuadrícula de γ y C fue constituida por los valores mostrados en las expresiones 6.2 y 6.3 respectivamente.

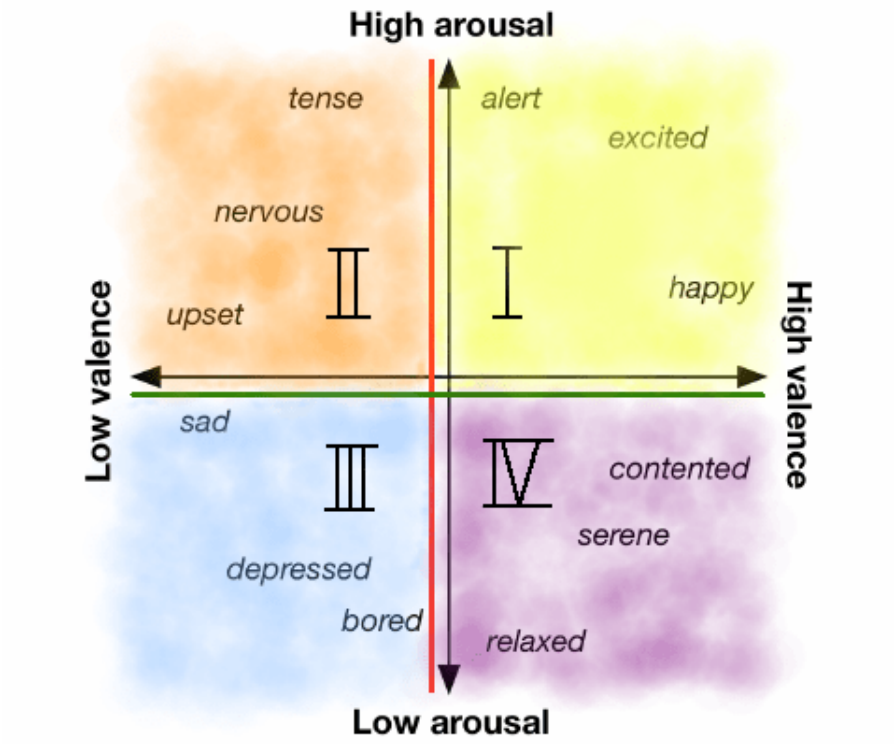

Figura 6.4: Separación de clases para SVC. La separación entre clases se realiza separando las emociones en 4 cuadrantes (I, II, III y IV), dónde la linea divisoria de cada emoción corresponde al promedio de las emociones en el conjunto de datos. Para la valencia se representa por la línea **roja** y para la activación por la línea **verde**

$$\Gamma = \{1 \cdot 10^{-17},\ 1 \cdot 10^{-16},\ 1 \cdot 10^{-15},\ 1 \cdot 10^{-14},\ 1 \cdot 10^{-13},\ 1 \cdot 10^{-12},\ 1 \cdot 10^{-11}\} \tag{6.2}$$

$$C = \{0{,}01,\ 0{,}1,\ 1,\ 10,\ 100,\ 1{,}000,\ 10{,}000,\ 100{,}000,\ 1{,}000{,}000\} \tag{6.3}$$

Para realizar lo descrito se utilizó *python*, específicamente la librería *pandas* para manejar los datos, *sklearn* para aplicar el algoritmo de ML y *matplotlib* para mostrar los resultados.

6.1.5. Elección del modelo a utilizar

Para la predicción de las emociones que inducen la música se utilizó una RNN-LSTM, ya que este tipo de aprendizaje de máquinas es adecuado para el aprendizaje sobre secuencias temporales, a diferencia de las NN *feed-forward Vanilla* y de las RNN *Vanilla* , como fue explicado en 2.3.4.

Primero se realiza un entrenamiento no supervisado DAE, para transformar la representación de las características y aportar así a la regularización del modelo. Luego, se elimina la última capa del modelo DAE y se le agregan capas RNN-LSTM y de Linearización, para predecir, con un enfoque multi-tarea, las variables dinámicas de valencia y activación. Es decir, se utilizó un entrenamiento semi-supervisado, como se muestra en la figura 6.5. Se implementaron diferentes variaciones de lo descrito, lo cual se mostrará en detalle en 6.1.6 y 6.1.7. Primero es necesario describir como se preprocesaron los datos.

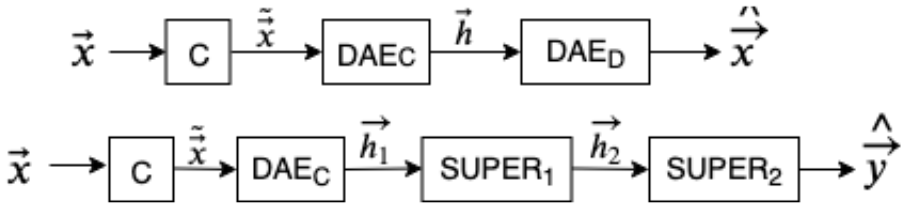

Figura 6.5: Entrenamiento semi-supervisado. El diagrama de flujo de la parte superior representa el entrenamiento no supervisado, mientras que el de abajo el supervisado.
C : Proceso de corrupción,
$\mathbf{DAE_C}$: Codificador DAE,
$\mathbf{DAE_D}$: Decodificador DAE,
$\mathbf{SUPER_1}$: Primera capa de entrenamiento supervisado,
$\mathbf{SUPER_2}$: Segunda capa de entrenamiento supervisado.

6.1.6. Autocodificador con eliminación de ruido (DAE)

Se probaron 2 estructuras de DAE diferentes, para utilizar luego el que generara una mejor predicción de las características de entrada. Para cada estructura se probó con diferentes

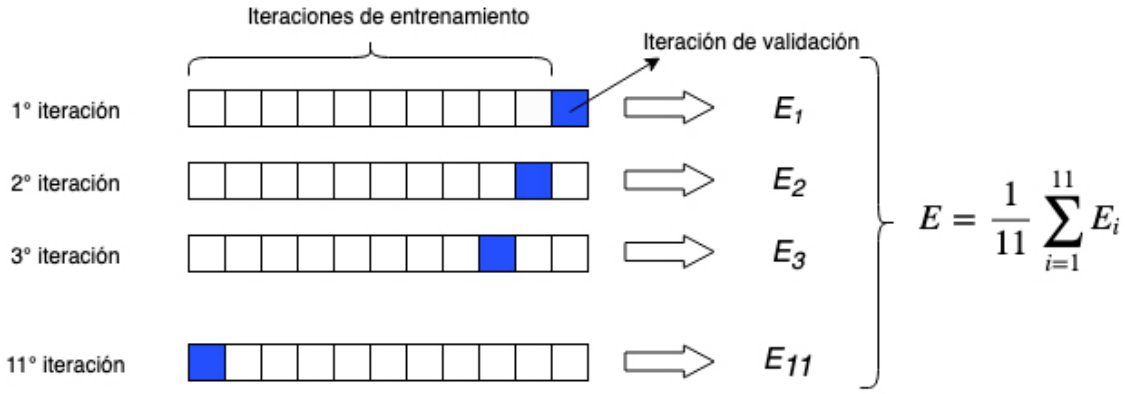

Figura 6.6: Validación cruzada de 11 separaciones.

hiperparámetros, que se muestran en la tabla 6.2, utilizando búsqueda por cuadrícula para su elección. En ambas estructura se utilizó la pérdida del error cuadrático medio y se probó con dos algoritmos de optimización: SGD con momento 0,9 y Adam con hiper-parámetros $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\varepsilon = 10^{-8}$. También, en ambas estructuras, se utilizó Validación cruzada de 11-separaciones para aumentar la data. Por otro lado, se probó, como métodos de corrupción de DAE, la eliminación de algunas características aleatorias y/o la agregación de ruido gaussiano a las características de entrada. Se describen a continuación las 2 estructuras.

6.1.6.1. Estado final como estado inicial (DAE V1)

En este caso, el estado final de la capa LSTM-RNN codificadora (LSTM_1) fue el estado inicial de la capa LSTM-RNN decodificadora (LSTM_2) y las entradas a LSTM_2 eran la predicción de las características desde la última hasta la primera característica, a excepción de la primera entrada a LSTM_2, que fue definida como un vector de ceros. La capa codificadora entonces tiene una estructura *many-to-one* y el decodificador *one-to-many*. Dicha configuración se muestra en la figura 6.7. De esta forma se esperaba que $\vec{h}_l^1$ tuviese toda la información que contenía la canción y fuera transformándose al pasar por LSTM_2 iterativamente. Notar que la secuencia se entregó desde el inicio hasta el final al codificador y al revés al decodificador, ya que se esperaba que el estado final del decodificador tuviese más información asociada a la última característica.

6.1.6.2. Configuración *many-to-many* (DAE V2)

En esta configuración cada entrada de características (de cada instante), tenía una salida al pasar por un bloque LSTM, luego de que pasa la canción completa, la característica transformada de cada instante pasa por una capa lineal, la que finalmente reconstruye la característica de entrada. Dicha configuración puede verse en la figura 6.8.

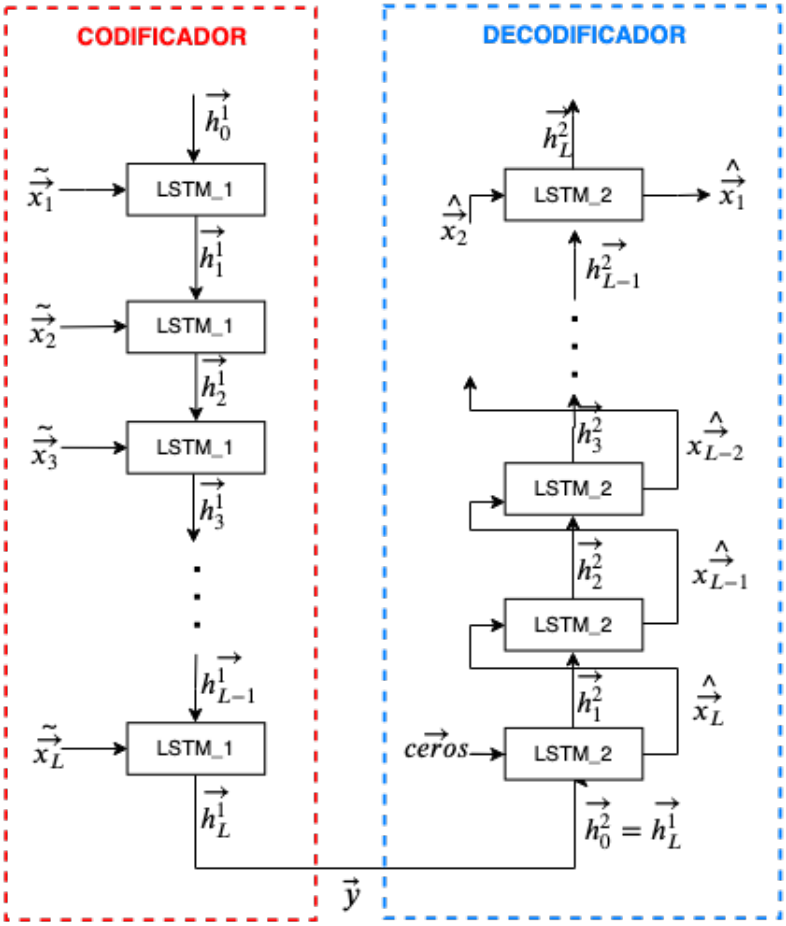

Figura 6.7: Estructura de DAE V1.
$\tilde{\vec{x}}_i$ **:** Característica de entrada i-ésima corrupta.
$\vec{h}_i^k$ **:** Estado de salida de red LSTM_k e iteración i-ésima.
$\vec{y}$ **:** Representación de características de entrada transformada.
$\hat{\vec{x}}_i$ **:** Predicción de características en cada instante i.

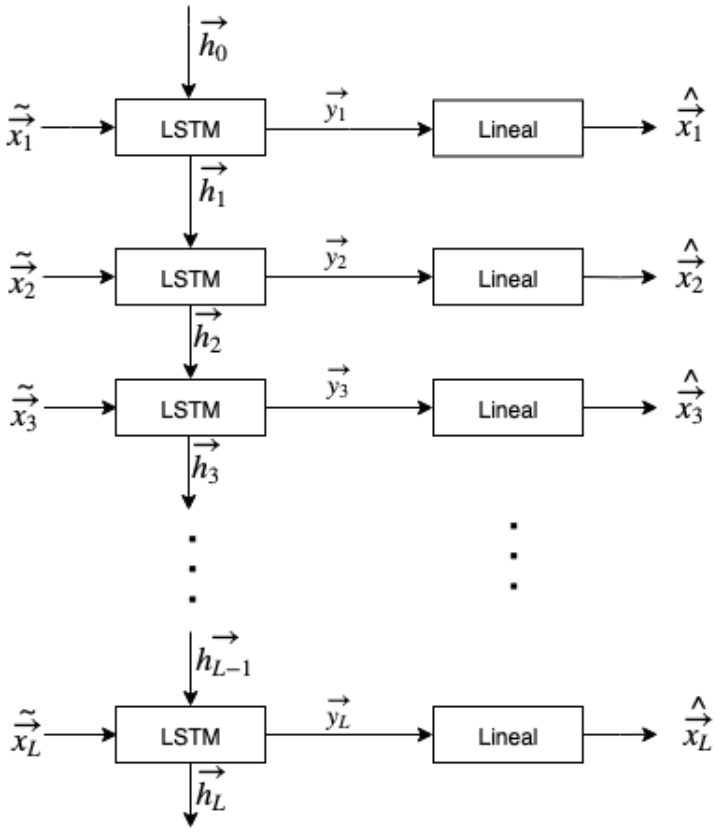

Figura 6.8: Estructura de DAE V2. $\tilde{\vec{x}}_i$ **:** Característica de entrada i-ésima corrupta. $\vec{h}_i$ **:** Estado de salida de iteración i-ésima. $\vec{y}_i$ **:** Representación de características de entrada transformada en instante i-ésimo. $\hat{\vec{x}}_i$ **:** Predicción de características en cada instante i.

Hiper-parámetros	Valores usados		
n	0	10	30
sd de ruido	0.3	1	0.1
Tamaño de mini-lote	5	10	20
Tasa de aprendizaje	5e-6	5e-5	5e-4
Unidades en capa oculta V1	200	1000	2000
Unidades en capa oculta V2	200	50	600

Tabla 6.2: Hiper-parámetros de DAE para cuadrícula: Hiper-parámetros utilizados en la búsqueda por cuadrícula para ambas versiones de DAE (V1 y V2). n representa el número de características que se eliminan en cada instante, cada vez que entran a la red LSTM, los demás hiper-parámetros se explican por si mismos.

6.1.7. Configuración de la red y entrenamiento de RNN-LSTM

La red se construyó con 3 capas ocultas. Primero se entrenó una red, de manera no supervisada utilizando *De-noising* LSTM-RNN *Autoencoders* (6.1.6), prediciendo las mismas características de entrada, generando otra representación de los datos en la capa oculta, garantizando una mayor robustez, para así prevenir el sobreajuste. Luego de determinar los parámetros del autocodificador se eliminó la capa de salida y se agregaron las otras 2 capas y la capa de salida con las variables de regresión (anotaciones). Se probó fijando los parámetros de la primera capa en lo que se obtuvo en el entrenamiento no supervisado (DAE) y también dejando dichos parámetros para que el entrenamiento supervisado siguiera optimizándolos. Se aplicó un enfoque multi-task de diferentes maneras: **1.** Sólo capa de salida se duplicó para valencia y activación, **2.** Ultimas dos capas se separaron para valencia y activación, dejando sólo la capa de DAE común a ambas. Se probó agregando ruido gaussiano en la entrada y luego de la primera capa, para aportar a la regularización. Como función de pérdida se utilizó MSE y como métodos de optimización se probó con Adam y con SGD con momento, utilizando los mismos parámetros que en DAE, recomendado por los autores de los métodos.El número de bloques de memoria de cada capa, la tasa de aprendizaje y la desviación estándar del ruido gaussiano fueron optimizados usando búsqueda por cuadrícula. Se utilizó Validación cruzada de 11 particiones, utilizando un coeficiente de paciencia de 25, es decir, si en 25 épocas el error de validación no mejora, entonces se detiene el algoritmo de entrenamiento y se pasa a realizar el test. Se hicieron diferentes pruebas, utilizando detención temprana en todos los subconjuntos posibles de entrenamientos, es decir: **1.** Sólo en DAE, **2.** En DAE y también en entrenamiento supervisado, **3.** Sólo en entrenamiento supervisado, y **4.** En ninguno de los dos. Las muestras de cada partición se entregan en orden aleatorio a la red. Los hiper-parámetros a incluir en la grilla para su elección se muestran en la tabla 6.3.

Hiper-parámetros	Valores usados		
n	0	10	30
sd de ruido	0.1	0.3	1
Tasa de aprendizaje	5e-6	5e-5	5e-4
Unidades en capa oculta 1	50	150	500
Unidades en capa oculta 2	10	25	100
Tamaño de mini-lote	5	10	20
Seguir optimizando DAE	Si	No	

Tabla 6.3: Hiper-parámetros de aprendizaje supervisado, utilizados en la búsqueda por cuadrícula.

6.2. Obtención de entropías

6.2.1. Materiales

Para este proyecto se utiliza el lenguaje *Python* y el *Toolkit Humdrum*. Este último, creado por David Huron en los 80 [59], sirve para obtener medidas estadísticas y descriptivas de archivos que contienen la información de piezas musicales en formato .krn. Este formato representa la música en un archivo de texto formado por un encabezado y un pie de página que dan información general sobre el archivo y un cuerpo que se divide en columnas (*spines*), donde cada *spin* representa un instrumento o una voz. Los elementos que contienen estas columnas son las notas, con sus duraciones y matices y pueden estar expresadas en diferentes lenguajes. Si son expresadas en *semits*, cada nota será un número, donde dicho número es la cantidad de semitonos de distancia de la nota con respecto a una nota de referencia. Este número puede ser positivo o negativo, dependiendo si la nota es más alta o más baja que la nota de referencia respectivamente. Un ejemplo de estos archivos, que utiliza *semits* para expresar sus elementos se muestra en las figuras 6.9 y 6.10, donde en la primera se puede apreciar primero el encabezado del archivo y luego los primeros dos compases y en la segunda, el último compás y el pie de página del archivo. Estas dos figuras son el inicio y término de la primera parte del opus 9, número 2, de Haydn. Toda la información necesaria para utilizar este Toolkit se encuentra en [59].

Se utilizaron canciones en formato **kern. Las canciones fueron: 38 Corales de Bach (Nacimiento Compositor: 1685), 48 Preludios y Fugas del Clavecín Bien Temperado de Bach, 33 Cuartetos de Barbershop, 12 Sonatas de Corelli Trio (Nacimiento Compositor: 1653), 23 sonatas de Handel trio (Nacimiento Compositor: 1685), 12 Telemann sonatas solistas para violín (Nacimiento Compositor: 1681), 5 Cuartetos de cuerdas de Haydn (Nacimiento Compositor: 1732) y 7 Cuartetos de cuerdas de Mozart (Nacimiento Compositor: 1756).

```
!!!COM: Haydn, Franz Joseph
!!!ONM: Opus 09, No. 2
!!!OTL:
!!!OMV: 1
!!!SCT: H /disk/dos2/musedata_2000/haydn/traut/quart/op09n2/stage1/01
!!!SCA: Thematisch-bibliographisches Werkevererzeichnis (A. van Hoboken)
!!!YOR: Trautwein Edition
!!!EED: Frances Bennion
!!!ENC: Frances Bennion
!!!CDT: 1732/3/31/-1809/5/31/
!!!OCY: Österreich
!!!YEC: Copyright (c) 1994, 2000 Center for Computer Assisted Research in the Humanities
!!!YEM: Rights to all derivative editions reserved.
!!!YEM: Refer to licensing agreement for further details.
!!!YEN: United States of America
**semits        **semits        **semits        **semits
*k[b-e-a-]      *k[b-e-a-]      *k[b-e-a-]      *k[b-e-a-]
*E-:    *E-:    *E-:    *E-:
*clefG  *clefG  *clefC3 *clefF4
*M4/4   *M4/4   *M4/4   *M4/4
*tb96   *tb96   *tb96   *tb96
=0-     =0-     =0-     =0-
r       r       r       r
3       r       r       r
=1      =1      =1      =1
7       -5 3    -2      -9
10      .       .       .
15      r       r       r
19      .       .       .
22      3 -5    -2      -9
19      .       .       .
15      r       r       r
10      .       .       .
```

Figura 6.9: Encabezado de un archivo ***.krn***.

```
3       3       3       -9
8       5       -4      -14
5       .       .       .
2       .       .       .
3       3       -5      -9
8       5       -7      -14
5       .       .       .
2       .       .       .
3       3       -5      -9
r       r       r       r
==      ==      ==      ==
*-      *-      *-      *-
!!!RWG1: This file contains only pitch and duration data.
!!!RWG2: Transposing instruments are not transposed.
!!!RWG3: Spines may have incompatible meters.
!!!RWG4: Durations of rests and tied notes may be interpreted.
!!!RWG: Key is interpreted using the Humdrum key tool.
!!!AFT: stage1
!!!END: 1997/04/15
!!format changes:
!!content changes:
!!public access: complete
!!!KEY: 2409001381
!!!EMD: Converted from MuseData to Humdrum Dec 16, 2000, by Andreas Kornstaedt using muse2kern.
```

Figura 6.10: Final de un archivo ***.krn***.

6.2.2. Procedimientos

Se utilizó la herramienta *Humdrum* para analizar las piezas. Para manejar las funciones de *Humdrum* se utilizó programación orientada a objetos en lenguaje *Python*. Luego se hicieron gráficos y tablas de interés utilizando *Excel* y aplicando herramientas estadísticas. Antes

de obtener las entropías, se limpian los datos eliminando elementos que no corresponden a notas, como acentos, matices y ligados, que también están representados por símbolos en los archivos **kern.

En cuanto a medidas de la información, no sólo la entropía en si misma (***entropía de primer orden***, ec. 4.1) toma un valor preponderante. Entre las otras que destacan y que se usan en este proyecto, están la ***entropía condicional*** (ecuación 4.6) y la ***entropía normalizada*** (ecuación 4.7). En 6.11 se muestra un diagrama del flujo de la información en este algoritmo.

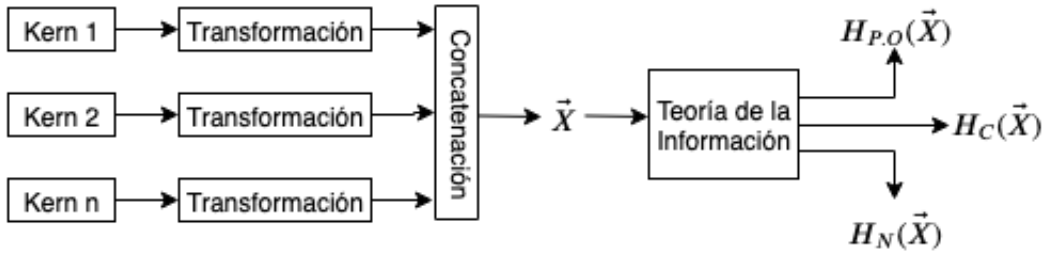

Figura 6.11: Diagrama de flujo para entropía. Muestra el flujo desde los archivos kern hasta las diferentes entropías obtenidas. Las transformaciones pueden arrojar los diferentes parámetros mencionados en 4. Las salidas finales son la entropía de primer orden, la entropía condicional y la entropía normalizada de dichas parametrizaciones.

En el caso de este trabajo, los eventos x e y se consideraron como eventos consecutivos, es decir, la entropía condicional sería la incertidumbre que existe con respecto a cada par de eventos consecutivos.

Por otro lado, la *entropía normalizada*, se define como la *entropía de primer orden* que efectivamente tiene un estilo de música, definido según un repertorio de un compositor, dividido por la *entropía máxima* que pudiese tener un evento aleatorio con el mismo alfabeto. Esta *entropía máxima* se da cuando cada evento tiene la misma posibilidad de suceder, es decir, si se repitieran todos los eventos la misma cantidad de veces en el transcurso de las obras del estilo en cuestión. Habiendo descrito lo anterior, nace la pregunta de cuál será la variable aleatoria para la cual se obtendrá la entropía, y dada la naturaleza de la música, es que existen diversas opciones. Podría ser el *tono* (frecuencia de la onda sonora), podría ser la duración de cada nota (tiempo desde que se hace sonar una nota hasta que se silencia), podría ser la relación entre notas adyacentes, la duración de la canción, los matices (diferentes formas de tocar una misma nota o un conjunto de notas) o muchos otros elementos.
En teoría musical, la armadura de una canción define un conjunto de tonos que serán usados en mayor medida en una parte, o en toda la canción. Es decir, define a que notas se le aplicaran sostenido o bemol, restringiendo así el abecedario (más probable) de una canción o de una parte de ella. Cada canción o pasaje de cada canción puede tener una armadura diferente, por lo que pierde el sentido analizar los tonos absolutos, ya que, por ejemplo, en una canción escrita en Do mayor, naturalmente existirán muchos Sol, Si, Fa y Do, dada las armonías comunes que se utilizan en esa escala. Pero es muy diferente que aparezca un Do cuando la armadura es Si menor.

Lo anterior condujo a tener que elegir ciertas características para el análisis, con respecto

a las cuales se obtuvieron las entropías. A continuación se describen las características utilizadas en este trabajo. En cada una primero se entrega una explicación conceptual y luego se identifica matemáticamente el alfabeto de la variable en cuestión.

- **Tono:** Es simplemente la nota que existe en cada momento. Estas serán representadas en cantidades de semitonos de distancia con respecto al Do central. Así un Re de la octava central tendría un valor de 2 y un sol de una octava más abajo tendría un valor de -5. *semits*.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = x_{0\mathrm{i}} \tag{6.4}$$

  El dominio de esta función ($x_{0\mathrm{i}}$) serán todas las notas que tiene un repertorio, al igual que el recorrido.

- **RCCSD:** Del inglés Registrally Centred Chromatic Scale Degree, son las notas representadas con respecto a la nota fundamental de la canción más cercana al Do central. Es decir, si una canción está escrita en Do mayor, sus semits serán los mismos que para el caso anterior, pero si está escrita en Re mayor, entonces un Re tomaría un valor 0 en semits.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = x_{0\mathrm{i}} - C(canción) \tag{6.5}$$

  Dónde C es es la la nota fundamental de cada canción del repertorio, más cercana al Do mayor de la octava central, expresada en semits. El dominio de esta función ($x_{0\mathrm{i}}$) serán todas las notas que tiene un repertorio y su recorrido serán dichas notas desplazadas en C unidades.

- **CSD:** Del inglés Chromatic Scale Degree. Este parámetro es igual que el anterior, pero el rango de posibles semits está entre 0 y 12. Para transformar los semits que se escapan de este rango, se aplica módulo 12, es decir si un semits es 15 en RCCSD, en CSD sería 3. Esto es equivalente a transponer todas las notas a la octava central, partiendo desde la fundamental de la armadura de la canción en cuestión.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = |x_{0\mathrm{i}} - C(canción)|_{12} \tag{6.6}$$

  El dominio de esta función ($x_{0\mathrm{i}}$) serán todas las notas que tiene un repertorio y su recorrido serán los números del 1 al 12 que resultan de la transformación del dominio.

- **Textura:** La textura se define en cuántas voces hay activas en cada línea del archivo .krn.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = len(l\acute{i}nea_{\mathrm{i}}) \tag{6.7}$$

Donde el lado izquierdo se refiere a la cantidad de elementos que hay en la linea i del archivo .krn, compuesto por todas las canciones de un repertorio concatenadas hacia abajo. El dominio de esta función es cada línea y su recorrido serán números naturales entre 1 y la cantidad máxima de voces activas en un instante.

- **Directed Melodic Intervals:** Representa los intervalos que existen entre cada evento y el siguiente, pudiendo estos ser negativos o positivos.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = x_{0\mathrm{i}} - x_{0\mathrm{i}-1} \tag{6.8}$$

  El dominio de esta función $(x_{0\mathrm{i}}, x_{0\mathrm{i}-1})$ serán todos los pares de notas consecutivos. Es importante recordar que como consecutivos se incluye la úlima nota de un spin y la primera del spin siguiente, ya que se concatenaron dichos spines. El recorido será Números enteros entre el intervalo mayor de los negativos y el mayor de los positivos.

- **Undirected Melodic Intervals:** Representa el valor absoluto de los intervalos que existen entre las notas del .krn.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = |x_{0\mathrm{i}} - x_{0\mathrm{i}-1}| \tag{6.9}$$

  El dominio de esta función $(x_{0\mathrm{i}}, x_{0\mathrm{i}-1})$ serán todos los pares de notas consecutivos. El recorrido serán los naturales menores que el máximo intervalo absoluto.

- **Contorno:** Cuando Directed Melodic Intervals toma un valor positivo, entonces Contorno toma un valor 1 y cuando el intervalo es negativo, Contorno toma el valor de -1. Cuando la nota se repite, Contorno toma valor 0. Representa las direcciones de los cambios.

  ◇ **Alfabeto:**

  $$x_{\mathrm{i}} = signo(x_{0\mathrm{i}} - x_{0\mathrm{i}-1}) \tag{6.10}$$

  La función signo arroja un -1 si el signo es negativo, un $+1$ si el signo es positivo o un 0 si el símbolo es un cero. El dominio de esta función $(x_{0\mathrm{i}}, x_{0\mathrm{i}-1})$ serán todos los pares de notas consecutivos. El recorrido será {-1,0,1}

Es importante notar que estos elementos aportan a la comprensión de la estructura melódica de las canciones, pero no de la estructura rítmica.

Para obtener la entropía en cada caso, primero se separó cada archivo en todas las voces que tiene y luego se concatenaron todas estas voces hacia abajo. En cada canción se hizo lo mismo y se concatenaron los resultados. Se obtuvo entonces un arreglo de todas las notas que aparecen en un repertorio y se pudieron aplicar las funciones correspondientes. En el caso particular de Textura, cada canción fue concatenada con la siguiente. Para obtener la entropía se hizo directamente con la herramienta infot de humdrum, la cual entrega la entropía según 4.1, es decir, obtiene la cantidad de veces que aparece cada nota en el repertorio y divide dicha cantidad por el total de notas que aparece, obteniendo así la probabilidad de que el compositor escoja cada nota. Este enfoque tiene un obstáculo, puesto que algunos elementos,

correspondiente a una voz y una línea tenían varios elementos, es decir, un elemento en la partitura correspondía a un acorde. Para solucionar esto y que las notas efectivamente formaran parte del repertorio, dichos acordes se dividieron en sus notas y se concatenaron hacia el lado en la posición donde aparecían en la voz respectiva.

6.2.3. Programa

El programa (proyecto.py), consta de 2 clases, una que maneja los archivos .krn (*class kernfile*) y otra que maneja conjuntos de archivos .krn que están en una misma carpeta (*class varioskernel*). En cada clase se definieron las funciones necesarias para obtener cada parámetro y además se generaron varias funciones externas a las clases que usan a las clases para obtener las medidas de la información. A su vez, todas estas funciones externas son llamadas una a una en el *main*. El código se muestra en el anexo A.1.

Capítulo 7

Resultados y Análisis

7.1. Predicción de emociones

7.1.1. Exploración de los Datos

Se generó el gráfico 7.1, constituido por diagramas de caja de 4 cuartiles con sus *outliers*, muestra el dominio en que se mueven algunas características dónde puede verse que los rangos son diferentes. Las últimas 4 características que se grafican están muy cercanas a cero, por lo que se hace un zoom a ellas que se muestra en el gráfico 7.2 y que también muestran rangos diferentes. Dada esta observación se hizo necesario normalizar esta datos datos, para que la red neuronal ponderara de igual manera las diferentes características.

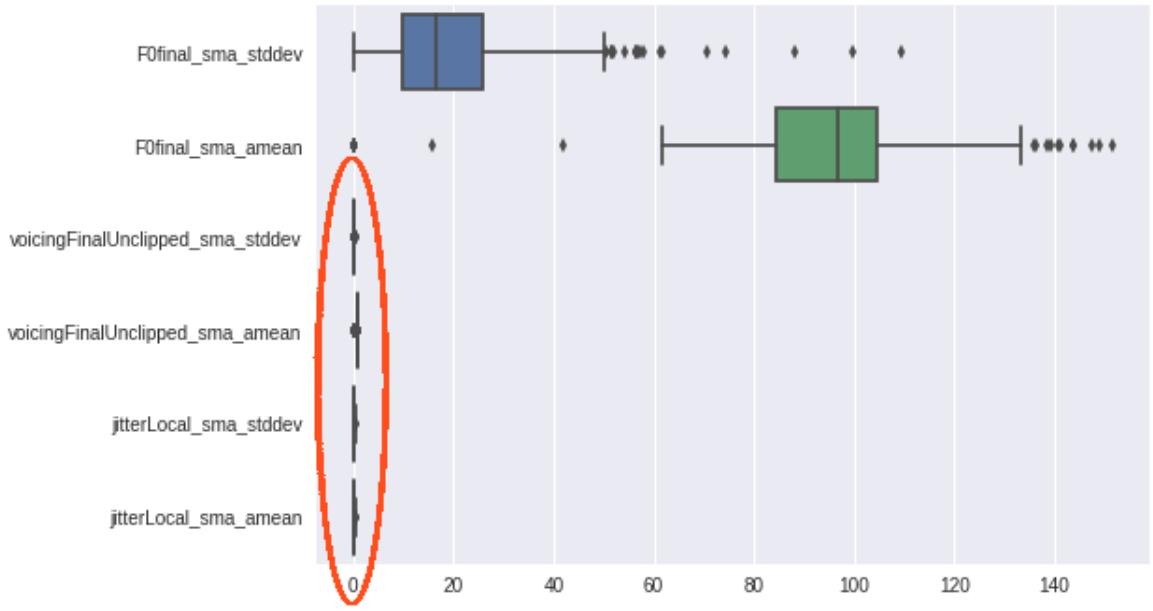

Figura 7.1: Diagramas de caja de ciertas características. Rango de valores de algunas características.

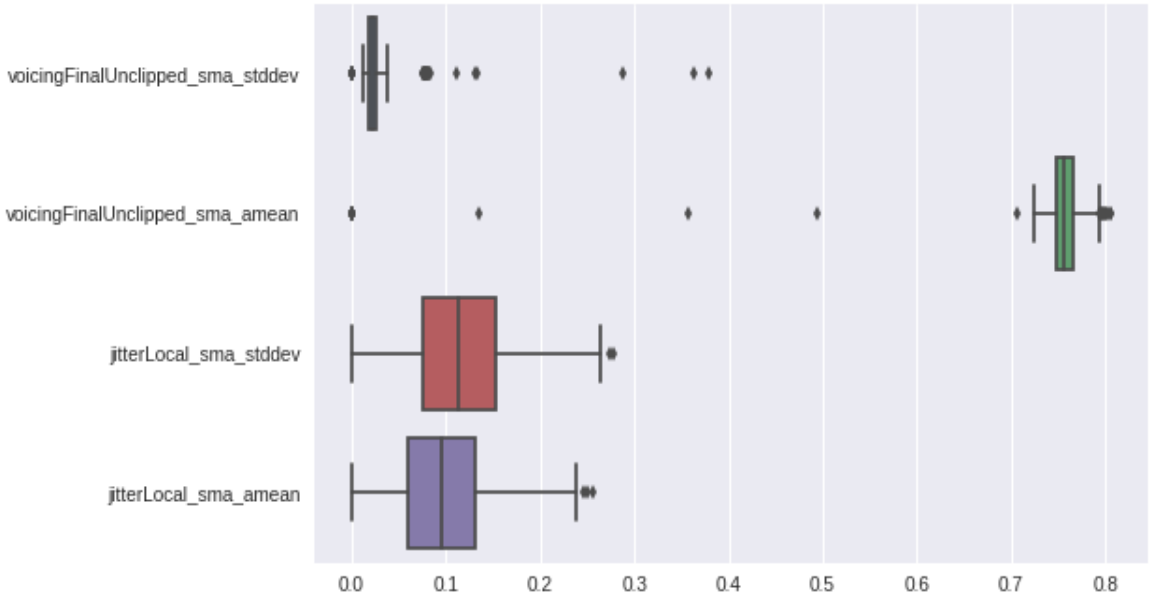

Figura 7.2: Subconjunto de diagramas de caja. Rango de valores de las características que no se pudieron apreciar adecuadamente en el gráfico 7.1. Este gráfico es un acercamiento al sector destacado en 7.1.

Acerca de los promedios de las anotaciones estáticas, en el gráfico (7.3) se puede ver una clara correlación, de hecho, la correlación de Pearson entre ambas variables es de 0.59. Luego se hizo lo mismo con las 58 canciones completas (7.4, donde queda al descubierto que no tienen el mismo comportamiento (r=-0.08), lo que podría dificultar la obtención de buenos resultados de prueba, ya que los datos de entrenamiento se comportan diferente a los de prueba, al menos en las anotaciones estáticas.

En las figuras 7.5 7.6 7.7 y 7.8 se muestran las desviaciones estándar y promedios de las excitaciones y valencias estáticas, donde puede verse que prácticamente no hay *outliers* y se concluye que no hay errores graves en la creación de la base de datos de valencia y activación estática.

7.1.2. Preprocesamiento

Antes de realizar el preprocesamiento indicado en 6.1.3, se encontró un problema en los datos: algunas anotaciones dinámicas por evaluador tenían más evaluadores en una categoría que en la otra (valencia y activación). Por esto, se tuvo que eliminar los evaluadores que estaban en una categoría pero no en la otra.

Los resultados de los 32 procesamientos diferentes se muestran en la tabla 7.1.

Figura 7.3: Relación entre promedios estáticos de valencia y activación para los 1744 extractos.

Figura 7.4: Relación entre promedios estáticos de valencia y activación para cada una de las 58 canciones completas.

ID	# de canciones	media de correlaciones de valencia	σ de correlaciones de valencia	media de correlaciones de activación	σ de correlaciones de activación
1	**544**	**0.789776**	**0.157993**	**0.823785**	**0.104555**
2	574	0.755391	0.254492	0.801992	0.181005
3	**544**	**0.789776**	**0.157993**	**0.823785**	**0.104555**
4	574	0.755391	0.254492	0.801992	0.181005
5	302	0.716283	0.212634	0.769342	0.117167
6	377	0.578469	0.464498	0.688074	0.341165
7	302	0.716283	0.212634	0.769342	0.117167
8	377	0.578469	0.464498	0.688074	0.341165

Figura 7.5: Desviaciones Estándar de Valencia para cada extracto.

Figura 7.6: Desviaciones Estándar de activación para cada extracto.

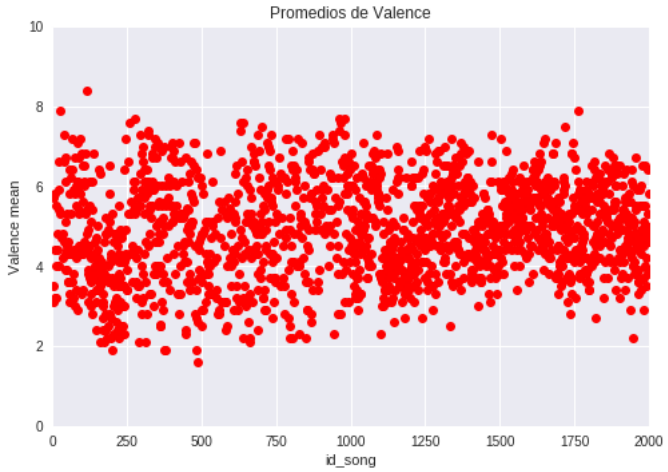

Figura 7.7: Promedios de Valencia para cada extracto.

Tabla 7.1: Información de correlaciones de cada pre-procesamiento.

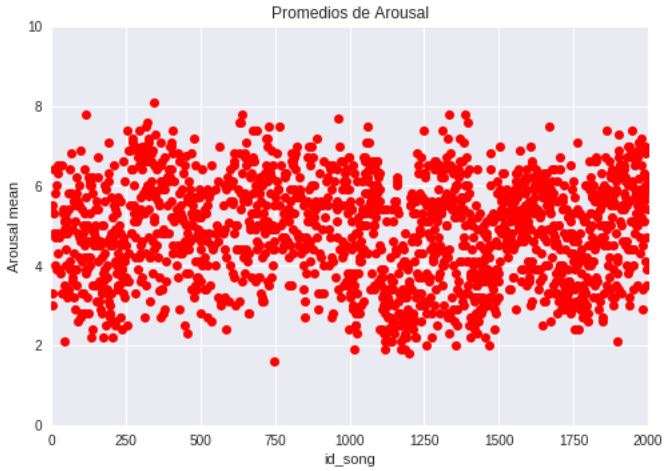

Figura 7.8: Promedios de activación para cada extracto.

Como puede verse en la tabla 7.1, las pruebas que tienen un mejor promedio de Alfa de Cronbach son la 1,3,17,19 y 25 (en **negrita** y **rojo**). Las combinaciones de decisiones en el preprocesamiento relativas a dichas pruebas se muestran en la tabla 7.2, dónde puede verse que fue mejor usar la definición de alfa de Cronbach de las correlaciones y borrar las canciones con alfa de Cronbach negativo. Las otras decisiones no presentan una posición definitiva. Al parecer la 1, 3, 17 y 19, resultan tener las mismas canciones (544 canciones para el conjunto de entrenamiento), lo que hace suponer que dichas canciones son las más adecuadas. Mientras que la 25 tiene 407, lo que es más cercano a las 431 de MediaEval y por este motivo también se utilizó. Se usó la prueba 1 y la prueba 25 (en **rojo**).

pba	borrarZeros	borrarPnan	corr	borrarAnan	borrarAneg
1	True	True	True	True	True
3	True	True	True	False	True
17	False	True	True	True	True
19	False	True	True	False	True
25	False	False	True	True	True

Tabla 7.2: La primera columna indica el número de la prueba. La segunda indica si se borraron o no las anotaciones que contenían solo ceros en una canción. La tercera muestra cuando se borraron las anotaciones que arrojaban un r de Pearson indefinido. La cuarta muestra cuando se usó la definición de Alfa de Cronbach de la matriz de correlación, en caso de ser *False* se usó la definición de la varianza de los ítems (ver 2.1.2). La quinta muestra si se borraron las canciones que arrojaban un Alfa de Cronbach indefinido. La sexta muestra si se eliminaron las canciones con Alfa de Cronbach negativo.

En resumen, el filtrado tuvo como resultado la disminución de 1744 a 544 y 407 extractos de canciones para el conjunto de entrenamiento. El promedio de los Alfas de Cronbach, en el caso de la prueba 1, para valencia fue 0.79, con una desviación estándar de 0.15, mientras que para la activación el promedio fue 0.82 y su desviación estándar fue 0.10. Para la prueba 2 el promedio de los alfas de Cronbach para valencia fue de 0.76, con una desviación estándar de 0.17, mientras que para la activación el promedio fue 0.80 y su desviación fue de 0.11. Estas canciones se separan en 11 partes, quedando 49 extractos para la prueba 1 y 37 para

la prueba 2. 10 partes se utilizaron para el entrenamiento y 1 para la validación, según 6.6.

7.1.3. Máquina de Soporte Vectorial

Los resultados que se obtuvieron se muestran en los gráficos de las figuras 7.9 y 7.10, dónde se expresa con mapas de calor lo obtenido como promedio de la validación cruzada con su respectivo test.

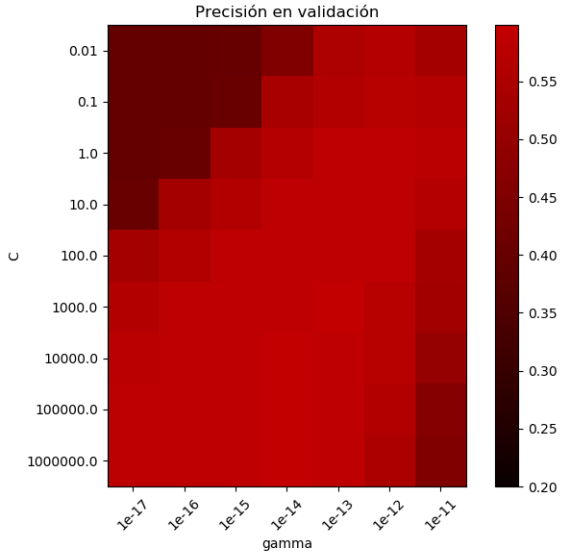

Figura 7.9: Mapa de calor para SVC utilizando todos los 1744 extractos para entrenamiento.

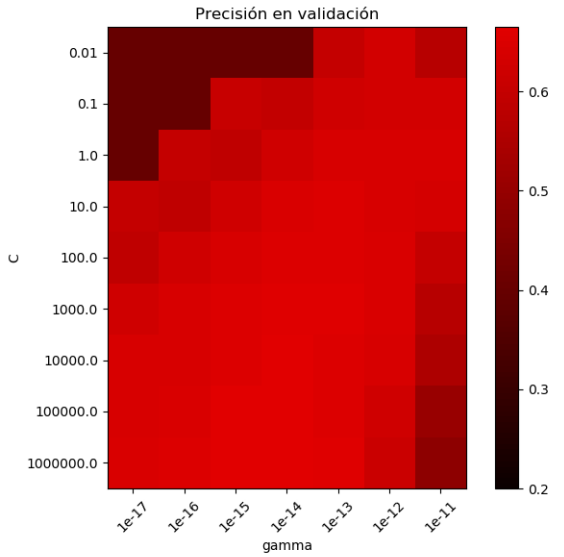

Figura 7.10: Mapa de calor para SVC descartando algunos extractos. Utilizando sólo los extractos cuya valencia y activación estática se diferenciaba del promedio de esta variable en todo el conjunto en al menos 0.2.

Los parámetros que mejor ajustaron el modelo, para el caso de utilizar todos los extractos fueron $C = 1{,}000{,}000$ y $\gamma = 1 \cdot 10^{-14}$ y se obtuvo una precisión de 0.6. Luego al usar estos parámetros en el conjunto de test (58 canciones completas), se obtuvo una precisión de 0.31. Para el caso de eliminar ciertos extractos del conjunto de entrenamiento los mejores parámetros fueron $C = 1{,}000{,}000$ y $\gamma = 1 \cdot 10^{-15}$, dónde se obtuvo una precisión de 0.67 y luego en el conjunto de test una precisión de 0.31.

Los resultados muestran que efectivamente mejora la precisión durante el entrenamiento al eliminar las canciones con emociones cercanas al promedio. Los resultados en la precisión cuando se utiliza el conjunto de test fueron bajos, lo que puede ser por dos motivos: 1. Existe un sobre-ajuste de los parámetros y 2. los datos de entrenamiento no se distribuían idénticamente, que es lo que se mostró en 7.1.1. En este caso, los valores de precisión entregados en el entrenamiento son sobre un subconjunto de test, dentro del conjunto de entrenamiento, por lo que un sobre-ajuste, como factor principal en el error, se descarta, aunque podría existir un leve sobre-ajuste, con resultados razonables en la precisión (0.6). Por lo tanto el problema de la no idéntica distribución de los datos de entrenamiento con respecto a los datos de test se identifica como el problema fundamental de la mala clasificación.

7.1.4. DAE

En esta sección se muestran y analizan los resultados del auto-codificador con eliminación de ruido. Separando dichos resultados en dos secciones, una para cada estructura utilizada.

7.1.4.1. DAE Versión 1

En esta versión, mostrada en 6.7 se obtuvo un muy mal desempeño. Los resultados de interés se resumen en los gráficos de las figura 7.11 y 7.12 y muestran que la pérdida alcanzó a ser un poco menor que uno, lo cuál no es lo esperado, ya que las variables de salida están normalizadas a media cero y desviación estándar 1, por lo que se espera un valor muy por debajo de 1 en el test.

7.1.4.2. DAE Versión 2

En esta versión, mostrada en 6.8 se obtuvo un desempeño adecuado. Los resultados de interés se resumen en los gráficos de las figura 7.13 y 7.14 y muestran que la pérdida alcanzó a ser cercana a 0.1, lo cuál si es lo esperado. Por ende, se utilizó uno de estos modelos para el aprendizaje supervisado.

Los hiper-parámetros que tuvieron mejor desempeño fueron:

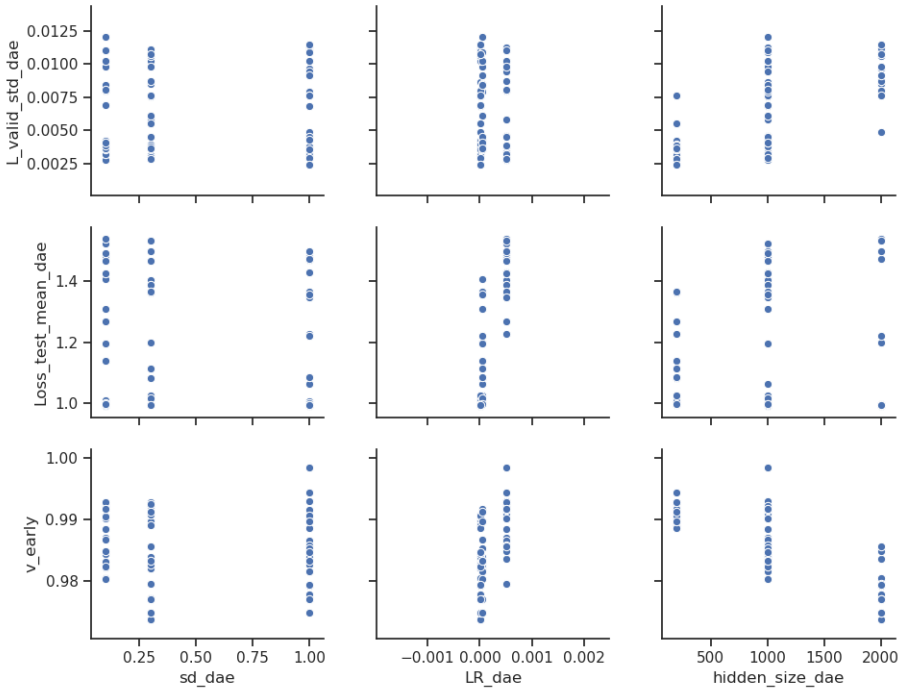

Figura 7.11: Relaciones entre hiper-parámetros (batch size y n) y pérdidas de validación y test en DAE V1.

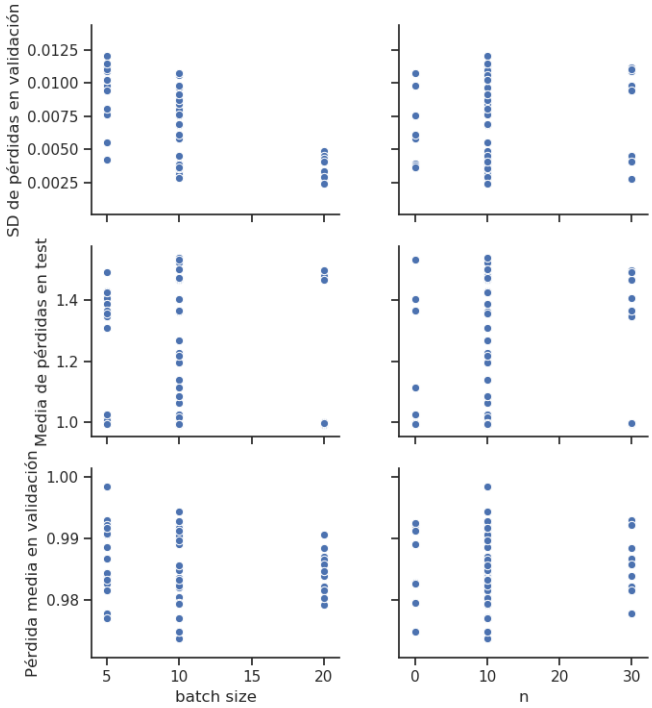

Figura 7.12: Relaciones entre hiper-parámetros (sd, lr y hidden size) y pérdidas de validación y test en DAE V1.

$$\boldsymbol{sd\ dae = 0{,}1}$$
$$\boldsymbol{n = 10}$$
$$\boldsymbol{hidden\ size = 600}$$
$$\boldsymbol{batch\ size = 10}$$
$$\boldsymbol{lr = 5 \cdot 10^{-5}}$$

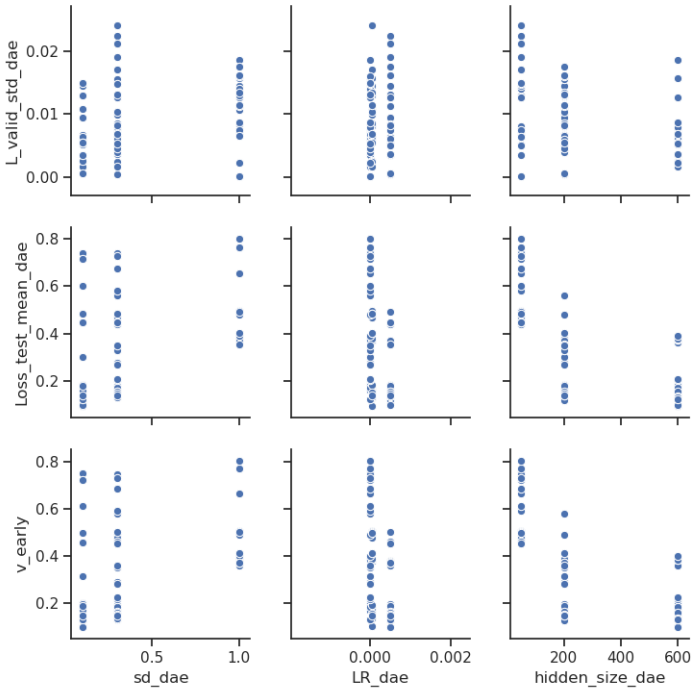

Figura 7.13: Relaciones entre hiper-parámetros (sd, lr y hidden size) y pérdidas de validación y test en DAE V2.

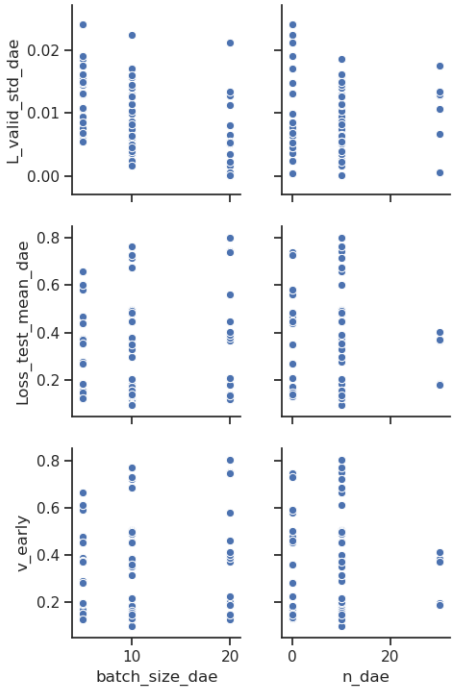

Figura 7.14: Relaciones entre hiper-parámetros (batch size y n) y pérdidas de validación y test en DAE V2.

Y sus resultados son:
Pérdida promedio de test: 0.096

Pérdida promedio en validación: 0.098
Cantidad de épocas en que se detuvo el algoritmo debido a Early-Stopping: 33
Tiempo de entrenamiento: 01:12:38

Aunque dicha configuración presentó los mejores resultados, hubo otras con resultados muy similares y con tiempos de entrenamiento mucho menores. Esto se muestra en la tabla 7.3, donde puede verse que la segunda fila es más conveniente, ya que es sólo una centésima mayor su error de test y el tiempo de entrenamiento se reduce a la mitad.

LR	L test	b size	stop ep	h size	n	sd	t_train	v_early
5e-05	0.0966083	10	33	600	10	0.1	1:12:38	0.0986348
0.0005	0.0976567	10	15	600	10	0.1	0:35:05	0.0957593
0.0005	0.118734	10	21	200	10	0.1	0:43:26	0.12407
0.0005	0.119596	20	26	200	10	0.1	0:51:25	0.125448
5e-06	0.122933	5	65	600	10	0.1	2:50:24	0.12693

Tabla 7.3: Hiper-parámetros y resultados de 5 mejores DAEs

A pesar de lo anterior, su usó el modelo de la primera linea de la tabla 7.3

7.1.5. RNN-LSTM

Los resultados del aprendizaje supervisado, utilizando DAE v1 como aprendizaje no supervisado no superaron un error de entrenamiento de 0.8 y esto puede ser debido a una mala implementación, ya que quizás se debió implementar DAE y RNN-LSTM de corrido, es decir, evaluar el error en la validación de las emociones para el aprendizaje de ambos modelos.

7.2. Obtención de entropías

Las medidas obtenidas se muestran en la tabla 7.4.

En los gráficos se usó la siguiente notación para los repertorios: Bach Chorales → BC, Bach WellTempered Clavier → BWC, Barbershop Quartets → BQ, Corelli Trio Sonatas → CTS, Handel Trio Sonatas → HTS, Telemann Violin Sonatas → TVS, Haydn String Quartets → HSQ y Mozart String Quartets → MSQ.

Los resultados de este experimento son similares con los de [79], existiendo diferencias en los cálculos individuales, pero teniendo siempre la misma relación de magnitud entre ellos. En el Anexo B.1 se muestran las comparaciones.

La entropía de primer orden para CSD, para cada repertorio, que se muestra en la figura 7.15, varía poco para este estudio. La desviación estándar para CSD primer orden fue de 0.06, que comparado con el promedio de las desviaciones, superior a 0.2, muestra claramente que

	Tono	*RCCSD*	*CSD*	*Textura*	*Directed Melodic Interval*	*Undirected Melodic Interval*	*Contorno*
Bach Chorales (BC)							
Primer orden	4,86	4,99	3,49	0,05	3,09	2,18	1,31
Condicional	2,87	2,95	2,88	0,05	2,63	2,19	1,34
Normalizada	0,90	0,89	0,97	0,05	0,66	0,56	0,83
Bach Well-Tempered Clavier (BWC)							
Primer orden	5,26	5,28	3,63	1,82	3,65	2,69	1,19
Condicional	3,62	3,63	3,28	0,70	2,96	2,51	1,18
Normalizada	0,91	0,90	0,95	0,52	0,67	0,58	0,51
Barbershop Quartets (BQ)							
Primer orden	4,44	4,60	3,54	0,58	3,07	2,48	1,57
Condicional	3,03	3,07	2,99	0,38	2,96	2,41	1,50
Normalizada	0,87	0,87	0,99	0,25	0,64	0,63	0,99
Corelli Trio Sonatas (CTS)							
Primer orden	5,12	5,36	3,51	1,43	3,42	2,50	1,22
Condicional	3,12	3,18	3,06	0,75	2,72	2,38	1,26
Normalizada	0,90	0,91	0,98	0,62	0,67	0,58	0,77
Handel Trio Sonatas (HTS)							
Primer orden	5,14	5,25	3,49	1,52	3,73	2,83	1,34
Condicional	3,44	3,50	3,14	0,75	3,01	2,62	1,29
Normalizada	0,88	0,88	0,94	0,59	0,69	0,61	0,67
Telemann Violin Sonatas (TVS)							
Primer orden	5,12	5,18	3,44	0,67	3,94	3,01	1,22
Condicional	3,52	3,57	3,19	0,53	3,15	2,77	1,18
Normalizada	0,89	0,89	0,96	0,42	0,73	0,67	0,77
Haydn String Quartets (HSQ)							
Primer orden	5,37	5,40	3,59	2,02	3,84	3,05	1,63
Condicional	3,60	3,62	3,20	1,12	2,93	2,49	1,33
Normalizada	0,89	0,88	0,97	0,53	0,64	0,59	0,82
Mozart String Quartets (MSQ)							
Primer orden	5,30	5,33	3,54	2,02	3,67	2,91	1,64
Condicional	3,51	3,52	3,14	1,22	2,83	2,41	1,33
Normalizada	0,89	0,88	0,96	0,58	0,65	0,61	0,82

Tabla 7.4: Resultados generales del procesamiento de la música.

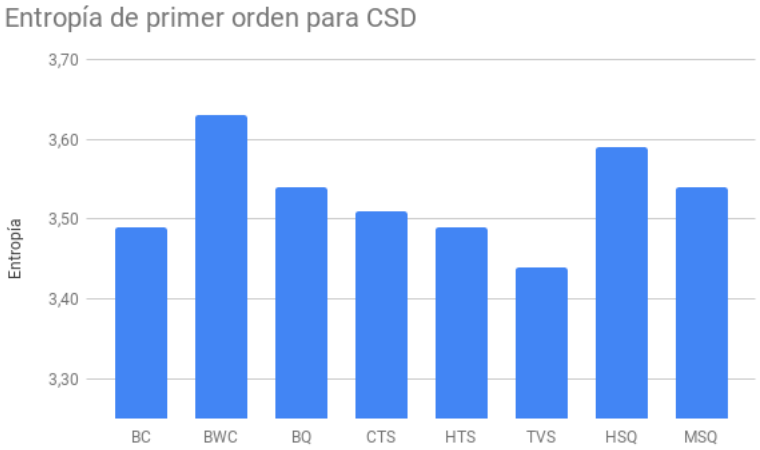

Figura 7.15: Entropía de Primer Orden para CSD.

CSD es más estable desde el punto de vista de las entropías. Esto es debido a su naturaleza, que por un lado las notas están relacionadas con respecto a la fundamental y por otro lado que todas las posibilidades de notas se reducen a una sola octava y reduciendo el alfabeto

produce menos variabilidad en las entropías. Esto indica que esta medida podría representar más bien el estilo de música en general de todas los repertorios, es decir, del sistema tonal utilizado en vez de los estilos en particular de cada repertorio.

De hecho, si vemos la comparación entre la cantidad de notas de cada frecuencia según CSD (figura 7.16), se puede apreciar que existe una alta correlación ($r=0.81$, $p<0.001$). Además, al considerar todos los repertorios, la distribución de frecuencias presenta una correlación alfa de Cronbach de 0.91

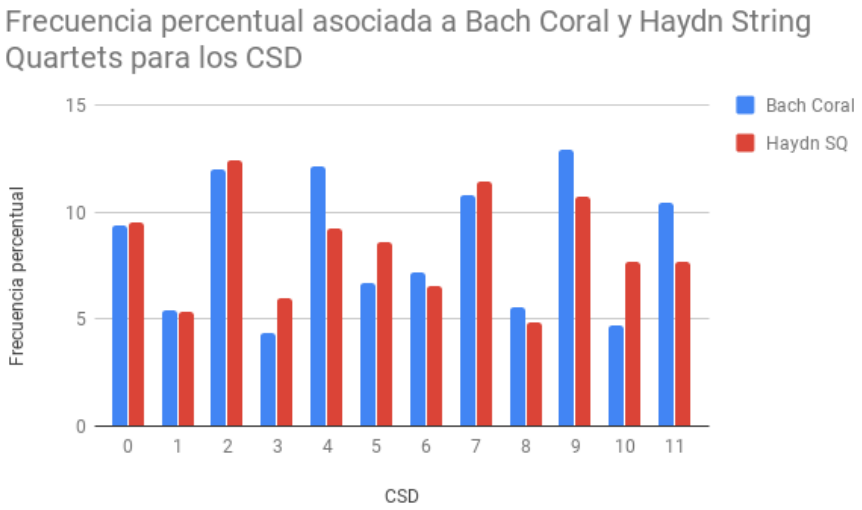

Figura 7.16: Comparación entre la Frecuencia Porcentual de apariciones de cada nota usando función CSD.

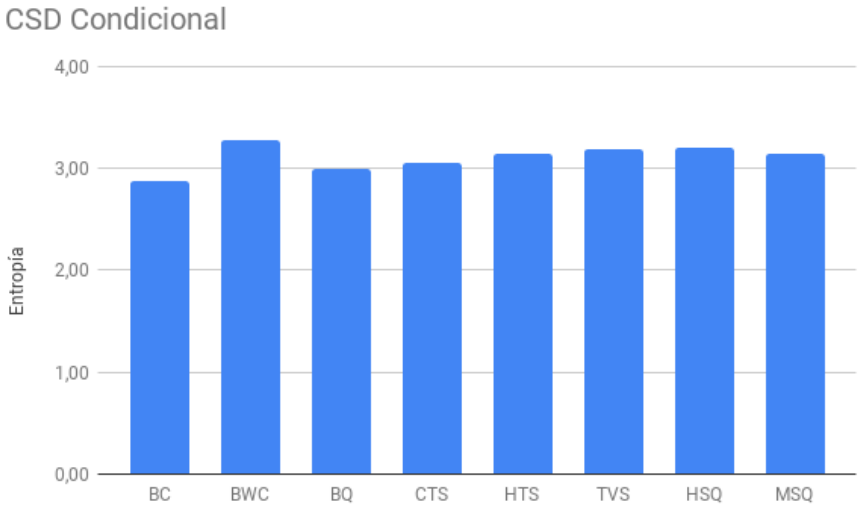

Figura 7.17: Entropía Condicional para CSD.

Los estilos pueden ser distinguidos de mejor manera por las entropías condicionales para CSD (figura 7.17). Puede verse que para los repertorios vocales (Barbershop Quartets y Bach

Chorales) se tiene una entropía condicional menor, esto puede deberse a que en general las melodías de voces son más suaves al ascender o descender y tienen un registro más acotado, por lo tanto existen menos posibilidades de intervalos y por ende menos elementos para el alfabeto, ya que cada elemento es un par de notas consecutivas. En cambio para el Clavecín bien Temperado o para las Sonatas de Violín de Telemann, los intervalos son mucho más libres, pudiendo saltar en intervalos mucho mayores y así su alfabeto junto con su entropía también son mayores. Este motivo también provoca que DMI, UMI , RCCSD y Tono tengan entropías condicionales menores para los repertorios vocales.

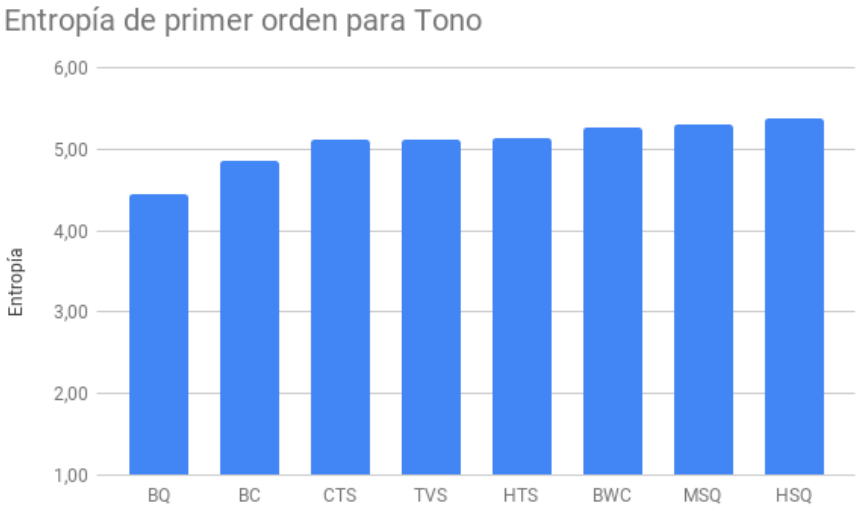

Figura 7.18: Entropía de Primer Orden para Tono.

La entropía de Tono refleja dos características básicas: la diversidad de armaduras que existen en una misma canción y el rango empleado. Ya que al existir más de una armadura en una misma canción, el algoritmo no detecta esos cambios de armadura y las diferentes notas se van tendiendo a ser igual de probables. Por otro lado, si el rango tonal es mayor en un repertorio, entonces tendremos un alfabeto más grande y por ende una mayor entropía (para distribuciones similares). En efecto, la entropía de Tonos para los String Quartet y para el clavecín bien temperado, son mayores (figura 7.18) porque sus instrumentos permiten movimientos sobre un rango mayor de tonalidades. En el gráfico 7.19 se muestran cuántas veces apareció cada nota cuando se aplica módulo 12 a la totalidad de las notas. Se puede ver que ambos compositores usan una distribución de notas similar. Esto hace de la medida CSD una medida que dice bastante, ya que con ella se puede ver las armonías, en general de las canciones. El hecho de que den similares es debido en parte a que la cantidad de canciones en tonos mayores y menores, para ambos repertorios, esta equiparada. Si hubiesen sido solo menores en un caso y solo mayores en el otro, las apariciones de las notas serían más diferentes.

En el gráfico 7.20 se pueden ver las relaciones entra la cantidad de veces que de una a otra nota se aumenta el tono y la cantidad de veces en que disminuye. Se puede ver, que la tendencia para los compositores de esa época es realizar más ascensiones, es decir subir más veces de intervalo y las bajadas las hacen más grandes cada una en promedio.

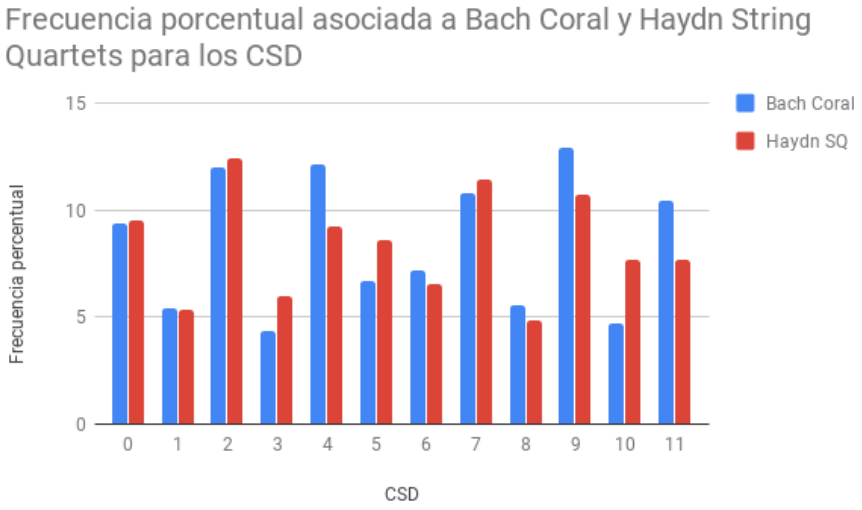

Figura 7.19: Relación entre las apariciones de las notas para Bach Chorales y Haydn String Quartets.

Figura 7.20: Razón entre la cantidad de intervalos positivos y negativos.

Se puede apreciar en el gráfico 7.21 que los Barbershops Quartet tienen un gran porcentaje de movimientos laterales, es decir, que repiten una misma nota, esto podría ser una característica importante para definir su estilo.

En el gráfico 7.22 podemos ver que la entropía es mayor para cuando se considera la dirección de los intervalos. Esto es evidente, ya que aumenta el alfabeto.

Figura 7.21: Porcentaje de eventos que repiten la nota anterior.

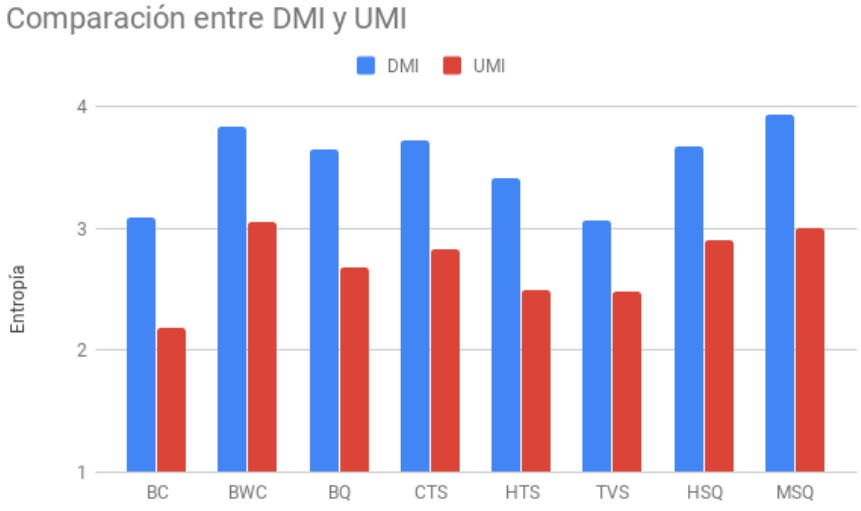

Figura 7.22: Entropía de Primer Orden para intervalos direccionados y no direccionados.

Capítulo 8

Experimento

En este capítulo se presenta el contexto necesario que fundamenta al experimento planteado, además del detalle del experimento en sí.

8.1. La Empatía en el Cerebro

La empatía está definida como la capacidad psicosocial natural de identificar la perspectiva cognitiva y experienciar el punto de vista emocional de otra persona de manera acertada [8, 18, 54, 53, 75, 123], además de tener respuestas emocionales adecuadas [10, 56], por lo que juega un rol crucial en las interacciones sociales. Existen muchas investigaciones sobre empatía que se basan en clasificaciones subjetivas [11] y en medidas auto-reportadas [81, 21, 33]. Claramente estos métodos son muy útiles para comprender la empatía de manera subjetiva, pero dichos estudios podrían estar influenciados por intenciones de aparentar de los individuos testeados [44]. Debido a esto se hace necesario investigar los mecanismos neurofisiológicos que están detrás de este fenómeno.

Existe una categoría de estímulos de gran importancia para primates y en particular para humanos, que es la generada por acciones realizadas por otros individuos. Si queremos sobrevivir, debemos entender las acciones de los otros. La imitación es fundamental para el aprendizaje y depende de la observación. Es más, sin entender las acciones, la organización social sería imposible. A diferencia de la mayoría de las especies, los humanos somos capaces de aprender por imitación y esta facultad está en la base de la cultura humana. El mecanismo neuro-fisiológico de las neuronas espejo parece jugar un rol fundamental en las acciones de entender e imitar la acción de otro [101]. El sistema de neuronas espejo (MNS) se basa en que durante la generación de movimiento, imaginación de movimiento y percepción de movimiento, se activan neuronas en común, aunque MNS no está localizado en región específica del cerebro [22, 24, 29, 85]. MNS no solo se activa al aprender acciones físicas, si no que también cognitivo sociales, como la mentalización y la empatía [61, 89].

Múltiples estudios anteriores han demostrado que la imitación es una respuesta automática en la que se involucra la empatía afectiva. Por otro lado la imitación es el principal motor

del aprendizaje [61, 72, 117, 121, 120, 131]). También se afirma que la imitación conduce a la empatía [122].

Ha sido estudiada la relación entre la empatía y los mecanismos neurofisiológicos, resultando que, además de lo expuesto sobre la MNS, los procesos empáticos generan una supresión de la onda mu. El ritmo mu, también llamado alfa percentral, alfa Rolándico o ritmo sensorimotor, es una onda cerebral que incluye frecuencias en el rango de 8-13 HZ ([95]), y ha sido observado en la corteza somatosensorial primaria [47, 93] (ver imagen 8.1). Existen varios tipos de ondas cerebrales. En la imagen 8.2 se observan las principales ondas cerebrales, la onda mu es similar a la onda alfa, pero difieren en morfología, distribución y reactividad. La supresión de la actividad mu se relaciona con fuertes flujos sanguíneos en la corteza somatosensorial [100] y ocurre cuando una acción es realizada, observada o imaginada [95]. La supresión de la onda mu es un indicador de la actividad de MNS [41, 88, 95] por lo que está relacionada con los procesos cognitivos y de movimiento [76]. Incluso, se ha comprobado que al realizar un test de Rorschach, cuando los estímulos de este provocan sensación de movimiento, los individuos presentan supresión de la onda mu [50], también se corrobora que estos estímulos están relacionados con la función cognitiva, empática y de cognición social [96]. También se sabe que mu se suprime cuando se ven objetos en movimiento, en comparación a cuando no se ven [66]. En niños de 18-30 meses se observó supresión de onda mu mientras observaban manipulación de objetos y cuando la imitaban, fue mayor la supresión en la zona central del cerebro mientras que fue menor en la parietal, además, la supresión se relacionó con la calidad de la imitación [132]. También existe supresión de la onda mu durante la imaginación de acciones observadas [90].

En base a una extensa cantidad de conocimiento sobre los mecanismos neurológicos se puede decir que la percepción del dolor en otros constituye un valioso paradigma para investigar lo que es neurológicamente la empatía [36]. Como podemos apreciar, en un estudio de magnetoencefalogarfía (MEG) se observaron cambios en la actividad de la corteza somatosensorial primaria provocados por la percepción de dolor en otros. Dicho cambio fue en la magnitud de la supresión de la onda mu, la que se toma como indicador de resonancia motora [26]. El dolor físico es activado por una red neuronal compleja: la Matriz del dolor [12, 30, 38, 99]). Esta matriz de dolor activa en parte la corteza somatosensorial; en el caso del dolor físico, la corteza cingulada anterior y la corteza insular; para la componente afectiva ([31, 98, 116]), además, estas activaciones pueden ser provocadas por respuestas vicarias de dolor que generan empatía ([55, 78, 130]). Estudios afirman que neuronas poseen propiedades que permiten intersubjetividad humana, haciéndonos capaces de empatizar y de reconocer el dolor en otros ([38, 45, 55]). También se afirma que las zonas cerebrales relacionadas con sentir dolor están parcialmente sobrelapadas con las zonas que se activan al ver dolor en otros ([78]).

Diferentes estudios clínicos demuestran que pacientes con desórdenes en los estados de ánimo muestran sus capacidades empáticas dañadas ([129]). Hay algunos estudios que por medio de evaluar la onda mu en personas con disfunciones mentales, comprueban que la supresión de la onda mu es menor en dichos sujetos. En sujetos con autismo existe supresión de onda mu cuando realizan una acción, pero no cuando la ven ([88]). En pacientes con esquizofrenia, mientras están siendo tratados con medicamentos, mejoran su evaluación en el test PANSS (Positive And Negative Syndrome Scale) y a medida que mejoran aumenta

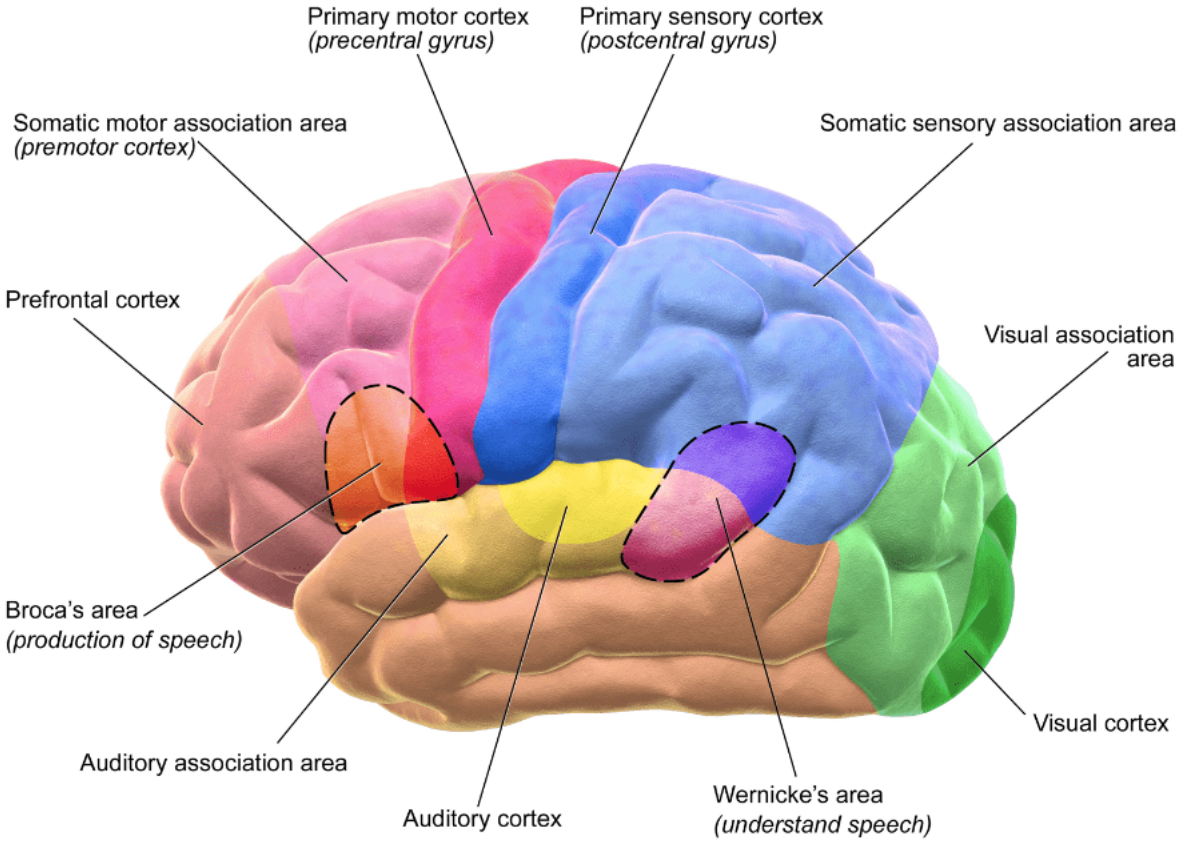

Figura 8.1: Partes de la corteza cerebral (adaptada de [124]).

la magnitud de la supresión de la onda mu en la corteza sensorimotora ([83, 84]). En otro estudio con sujetos con BPD (Borderline Personality Disorders), quienes presentan disfunción interpersonal y disfunción para empatizar, se mostró que no existe una correlación de la supresión de la onda mu con respecto a su evaluación en el test de empatía IRI (Interpersonal Reactivity Index), por otro lado, los sujetos de control si presentan una correlación entre estas variables, lo que indica que una mayor capacidad empática se relaciona con la supresión de la onda mu de manera directa en sujetos normales, pero no en sujetos con BPD ([80]).

Por otro lado, se ha encontrado que el estado de ánimo de los observadores influencian en gran medida su imitación ([71, 113]). Likowski et al ([113]) realizó electromiografías de las reacciones de los músculos faciales cuando los participantes observaban caras con expresiones de alegría, tristeza, enojo y de neutralidad anímica después de la inducción de estados de ánimo de felicidad y tristeza usando videos. Los resultados muestran que una inducción de estado de ánimo positivo conduce a una reacción facial coincidente con todos los estados de ánimo de las caras observadas, mientras que un estado de ánimo negativo inducido conduce a una reducción de la reacción facial, sugiriendo que, comparado con el estado de ánimo positivo, el negativo suprime la imitación facial.

Basándose en los argumentos mencionados se podría predecir que la componente motora de la empatía hacia el dolor sería suprimida cuando los observadores estuviesen en un estado

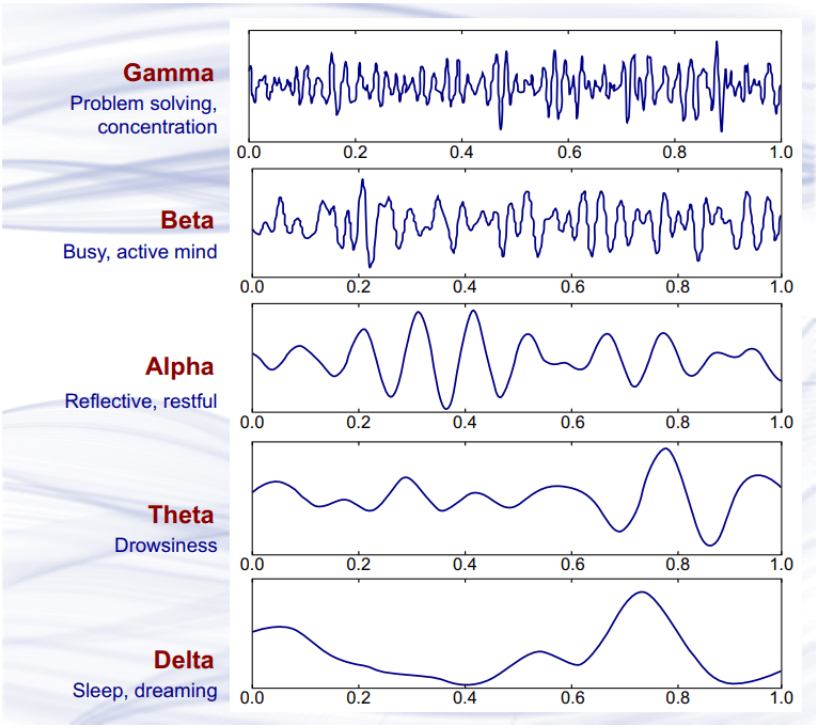

Figura 8.2: Comparación de frecuencias de ondas cerebrales (adaptada de [2]).

de ánimo negativo y sería aumentada cuando estuviesen en un estado de ánimo positivo. Esto es demostrado por Xiang et al ([74]), ya que su estudio arroja como resultados que cuando los individuos son inducidos a un estado de ánimo negativo la supresión de la onda mu es menor que cuando son inducidos a un estado de ánimo positivo.

En síntesis, se plantea que la música, al provocar diferentes estados de ánimo o emociones ([65]), modularía de diferente manera la percepción del dolor en otros, lo que se podría observar midiendo la supresión de la onda mu. Junto con esto, es posible que la complejidad de la música y conjunción con el entendimiento musical que tenga el sujeto testeado, también module la empatía del sujeto al percibir dolor en otros. Esto es lo que se quiere observar con el experimento que se propone en las líneas siguientes. Un simple diagrama para visualizar el raciocinio se muestra en 8.3.

8.1.1. Cuestionarios para medir la Empatía

Existen muchos cuestionarios que dicen servir para evaluar la empatía. A continuación se describirán algunos de los más citados por trabajos de investigación.

8.1.1.1. Interpersonal Reactivity Index(IRI)

Davis en 1980 diseñó este test [32] con dos objetivos principales: que fuera fácil de aplicar y que capture diferencias en reacciones cognitivas y emocionales. Su investigación resultó 28

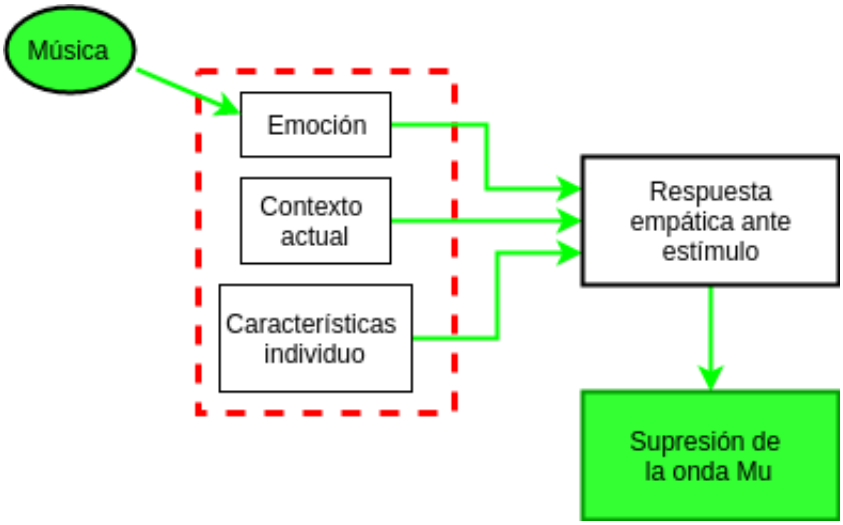

Figura 8.3: Flujo de implicancias en teorización de experimento.

elementos evaluados en una escala de Likert de 5 puntos desde 0 hasta 4. Los 28 elementos se pueden subdividir en 4 categorías: Fantasy Scale (FS), Perspective-Taking (PT), Empathic Concern (EC), Personal Distress (PD). EC se refiere a sentimientos de compasión y preocupación por otros. PT evalúa intentos no planeados para adoptar puntos de vista de otros. FS describe la probabilidad de que una persona se identifique con un personaje ficticio. PD indica el grado en que un individuo siente inquietud preocupación cuando es expuesto a experiencias negativas de otro. Esta escala está ampliamente validad y es una de las medidas de empatía más usadas.

8.1.1.2. CPI Q-Sort

CPI Q-Sort ([19]) es un instrumento lingüístico cuyo objetivo es permitir la descripción completa, en términos psicodinámicos contemporáneos, de la personalidad de un individuo en una forma adecuada para la comparación y el análisis cuantitativo. Este test tiene 100 items de verdadero o falso.

8.1.1.3. EETS - Emotional Empathy Tendency Scale

El EETS ([81]) es un test diseñado para evaluar la empatía emocional. Consta de 33 preguntas tipo escala de Likert de 9 puntos (de -4 a 4).

Existen diversas maneras de posicionar los electrodos, a continuación se muestra la del Sistema Internacional 10-20 (figura 8.4): El EEG es el método más práctico y replicable para detectar activación de MNS y supresión de onda mu en la corteza somatosensorial ([110, 76])

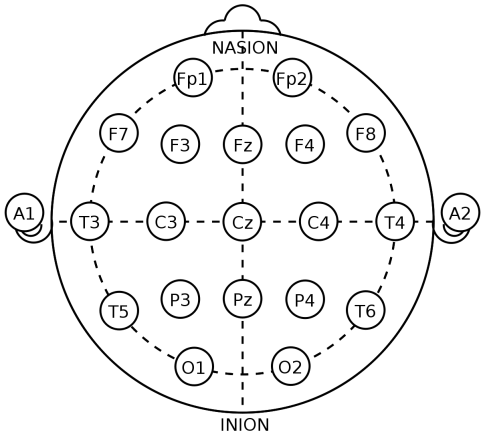

Figura 8.4: Posicionamiento de Electrodos según Sistema Internacional 10-20.

8.1.2. Cuestionario relacionados con la música

Para tener un análisis más robusto se propone incluir un cuestionario acerca de la relación del individuo con la música, y en particular con las piezas escuchadas. Se define un esbozo de dicho cuestionario a continuación:

- ¿Está usted familiarizado con la música de alguno de estos repertorios y/o compositores?:
  - ◇ Corales de Bach
  - ◇ Clavecín bien temperado de Bach
  - ◇ Cuartetos de Barbershop
  - ◇ Corelli trios sonatas
  - ◇ Handel trio sonatas
  - ◇ Telemann: violín sonatas
  - ◇ Haydn: cuarteto de cuerdas
  - ◇ Mozart: cuarteto de cuerdas
- ¿Escucha usted habitualmente y/o le gusta alguno de estos estilos de música?:
  - ◇ Rock
  - ◇ Pop
  - ◇ Soul
  - ◇ Blues

- ◇ Electrónica
- ◇ Clásica
- ◇ Hip-hop
- ◇ Internacional
- ◇ Experimental
- ◇ Folclórica
- ◇ Jazz
- ◇ Country
- ◇ Reggae
- ◇ Rap

- ¿Qué tanto le agradaron los extractos de música que acaba de escuchar? (Item de *Likert*)
- ¿Qué tan instruido en música está? (Item de *Likert*

8.2. Mediciones Experimentales

8.2.1. Descripción General

Se reclutarán a 10 personas de edades entre 18 y 30 años, intentando equiparar los géneros. Para obtener resultados estadísticamente confiables sería ideal que fuesen más personas, pero los alcances de este trabajo plantean al experimento como un piloto para evaluar resultados parciales y evaluar el experimento en si mismo.
Antes del experimento se realizará el cuestionario IRI ([32, 34]), para conocer su nivel de empatía. Este cuestionario consta de 28 afirmaciones evaluados en una escala Likert de 5 puntos que va desde "no me describe bien"hasta "me describe muy bien". Este es subdividido en cuatro subescalas con el objetivo de reportar diferentes dimensiones de la empatía emocional y cognitiva, como se explicó en 8.1.1.1. Este test es el más extensamente validado y utilizado frecuentemente en estudios neurológicos de la empatía ([62, 48, 92, 7, 27]). También se le realizará el cuestionario concerniente a su experiencia y gustos musicales (8.1.2. Un diagrama de flujo del experimento se muestra en la figura 8.5

Los participantes se inscribirán después de leer la siguiente descripción del experimento: "Si decide participar, usará una gorra con electrodos que registran su actividad cerebral mientras escucha música. La sesión experimental durará aproximadamente 40 minutos"; en la Tabla 8.1 se detalla la estimación de tiempos. Luego firmarán un consentimiento informado

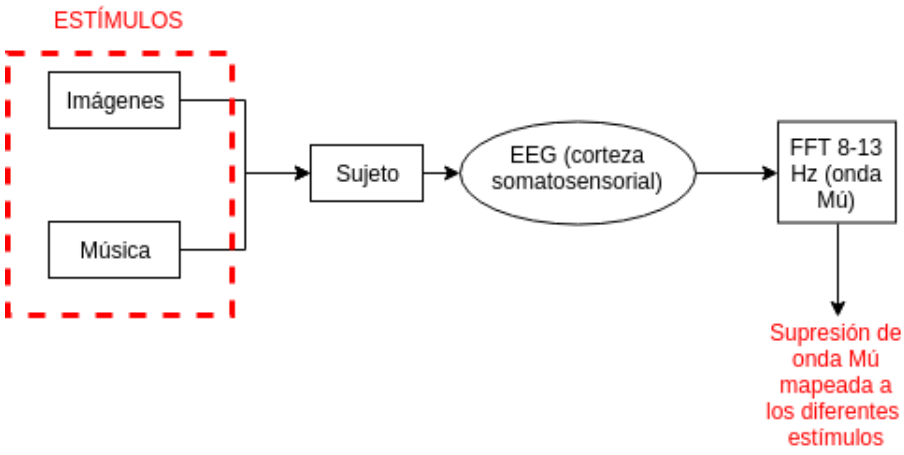

Figura 8.5: Diagrama de flujo del experimento

visado por el comité de ética de la Universidad Diego Portales, lugar donde se realizarán los experimentos. El experimento cumplirá con la Declaración de Helsinsky de 1964.

Actividad	**Tiempo [min]**
Cuestionario IRI	15
Cuestionario música	15
Preparación experimento	5
Experimento	5
Margen de error	5
Total	**45**

Tabla 8.1: Estimación de duraciones en actividad experimental.

8.2.2. Materiales

Habrán dos tipos de música en el experimento, música clásica que está en archivos **kern, formato del cuál es fácil extraer medidas de teoría de la información como la entropía, como se explicará más adelante. Entre esta música serán seleccionados 10 extractos con duración de 40 segundos, 5 de repertorios que tengan una alta entropía en relación a los otros 5.
El otro tipo de música será música contemporánea, de todo tipo, de la base de datos DEAM, la cuál fue creada durante los años 2013, 2014 y 2015 en el contexto del concurso de clasificación según emociones, esta base de datos incluye información sobre el grado de excitación y la valencia de las canciones cada medio segundo, como se establece en el modelo Circumplex de afectividad, las cuales serán caracterizadas según el estado de ánimo con el que están relacionadas, como se detalla más adelante. De este tipo de música se tomarán 10 extractos, también con duración de 40 segundos, con diferentes estados de ánimo asociados, 5 con valencia alta y 5 con valencia baja, todos con excitación alta.
Se utilizarán imágenes que tienen contenido de dolor y otras imágenes que no lo tienen, extraídas de [44]. Serán 410 imágenes, 205 de dolor y 205 de no dolor (imágenes pueden repetirse). Estas figuras son de manos miradas en primera persona como se muestra en la figura 8.6.

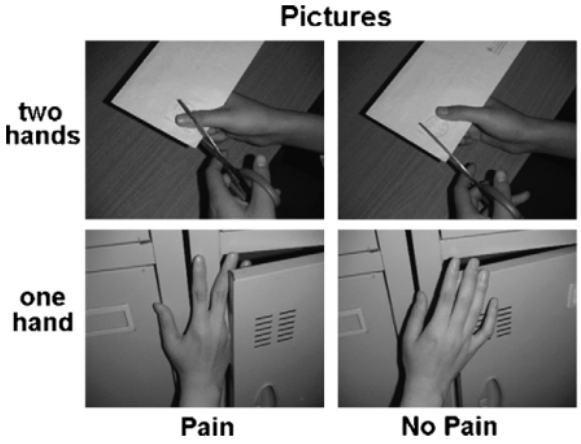

Figura 8.6: Imágenes de dolor y no dolor.

8.2.3. Procedimiento

Primero el sujeto a 150 centímetros de una pantalla se le aplicará el bloque A del experimento, luego se le aplicará el bloque B_0 y a continuación, sin detener la música, el bloque B se repetirá 20 veces (una vez para cada extracto de música). Los bloque A y B se muestran en la figura 8.7, donde se explicitan sólo 4 imágenes por bloque, por un tema de espacio en la imagen, siendo en realidad 10 para el bloque A y 20 para el bloque B. El experimento de EEG dura en total 1140 segundos (19 minutos).

- **Bloque A:** Se mostrará una imagen negra por 15 segundos, a modo de registro de control. Luego se intercalarán 10 imágenes de las mencionadas (1 segundos cada imagen) con 9 intervalos de pantallas negras (1 s cada pantalla negra), para finalizar esa etapa se mostrarán 5 segundos de pantalla negra.

- **Bloque B0** Consiste en 10 segundos de pantalla negra con el inicio del extracto musical.

- **Bloque B:** Parte con 5 segundos de pantalla en negro con música, luego de lo cuál comenzará otra serie de 20 imágenes intercaladas con 19 pantallas en negro con las mismas duraciones que para el bloque anterior pero con la música sonando. En los útlimos 5 segundos tendrá que responder si acostumbra a escuchar el tipo de música que escucho y si había escuchado antes esa música en particular.

Al final de cada canción el sujeto deberá responder que estado de valencia y de excitación le produjo, en una escala de Likert. Además de si le agradó o no la música, en la misma escala y si conocía o no la canción.

La data de la música y las imágenes es sincronizada y marcada en el registro del EEG para así poder realizar la correlación dinámica adecuadamente.

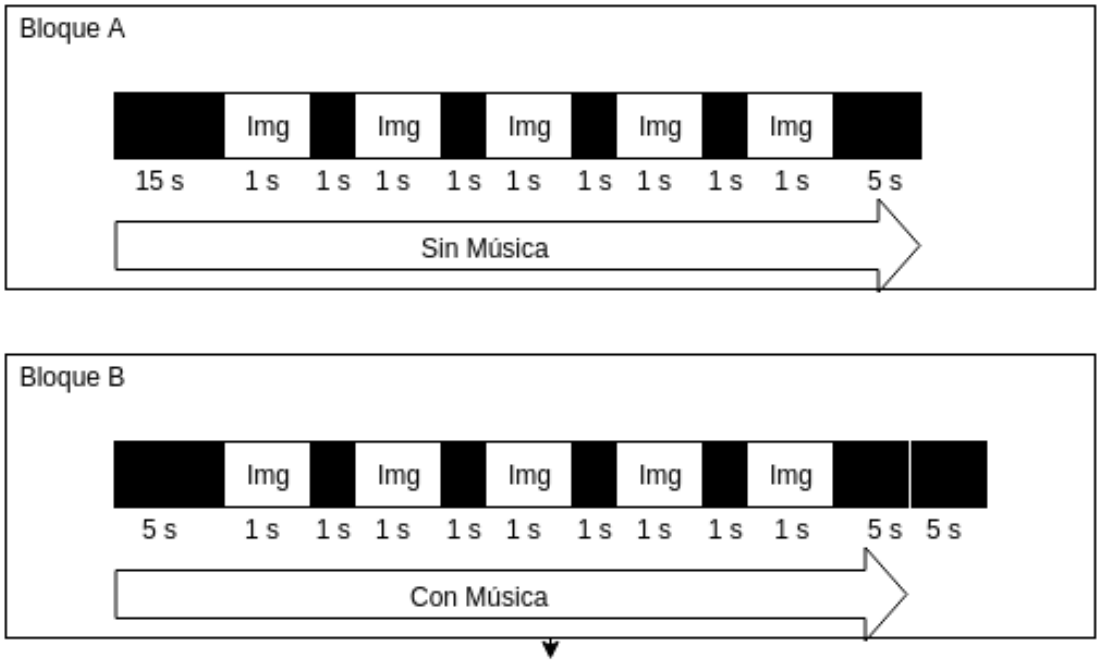

Figura 8.7: Secuencia del experimento.

8.3. Preprocesamiento y grabación de EEG

El electroencefalograma (EEG) fue grabado continuamente a partir de 64 canales de electrodo activo montados en una cubierta ampliada, personalizada y elástica (Brain Products ActiCap) utilizando un filtro paso banda de 0.01–100 Hz a una frecuencia de muestreo de 500 Hz. Los 64 canales se encuentran distribuidos de forma homogénea por toda la cabeza. Las impedancias de todos los canales se mantuvieron por debajo de 15 $k\Omega$.

Los datos de EEG se exportarán a MATLAB (MathWorks) y se procesarán utilizando el paquete de software EEGLAB y los scripts personalizados. Se filtran las frecuencias menores que 8 Hz y mayores que 13 Hz utilizando un filtrado lineal, se tomarán muestras a 250 Hz. Para cada participante, se implementó un procedimiento para la eliminación de artefactos, basado en un enfoque de análisis de componentes independientes (ICA) que se ha establecido previamente ([35],[42],[112, 109]) para obtener datos de EEG minimizando los artefactos oculares/biofísicos. Primero se rechazan visualmente las partes inadecuadas de los datos del EEG continuos, como ruidos no neuronales (desviaciones, parpadeos excesivos). En segundo lugar, separamos los datos restantes en períodos de 1500 ms, que abarcaban desde 500 ms antes del estímulo hasta 1000 ms después del inicio del estímulo. Luego, se realiza un infomax temporal ICA ([13]). Y finalmente, se eliminan los datos provenientes de artefactos conocidos (como, parpadeos y latidos cardíacos).

8.4. Análisis de datos de ERD

Para el análisis se usan los datos referentes a la corteza somatosensorial. Para analizar la supresión de la onda mu (o Desincronización Relacionada a Eventos-ERD) en el EEG se utiliza FFT (Fast Fourier Transform), como se realiza en [88, 94, 118]. También se apilcará una descomposición wavelet de Morlet.

Capítulo 9

Conclusiones

Este trabajo fue de mucho valor en cuanto al aprendizaje de su autor. Se logró diseñar diferentes algoritmos de ML y utilizar de manera práctica la Teoría de la Información. En particular, se rescata el aprendizaje en cuanto a estrategias de regularización y específicamente sobre el algoritmo RNN-LSTM, además de la comprensión teórica de la TI. Otro aprendizaje esencial fue el manejo de todas las librerías utilizadas. Todo esto forma una base sólida para seguir avanzando en estos temas de manera más fluida.

Indiferente a los resultados obtenidos de los algoritmos, se considera que el diseño y metodología planteada para solucionar los problemas definidos al comienzo, representan un avance en el conocimiento, en particular en la aplicación de herramientas ingenieriles al área de la Psicología. Además, los resultados que cumplen las expectativas de predicción son un reflejo del trabajo en cuanto a investigación, diseño e implementación de las herramientas utilizadas.

Lo incluido en Marco Teórico logró las expectativas pedagógicas, siendo este apartado ampliamente recomendable para estudiantes que quieran comprender a cabalidad los algoritmos de ML presentados y someramente lo usado con respecto a Teoría de la información, constituyendo así una excelente herramienta introductoria a las disciplinas mencionadas.

Es importante mencionar que la clasificación de la música según emociones es útil en el experimento, puesto que se podría contrastar los casos en que el algoritmo predice un resultado diferente a lo anotado por los encuestados. De este contraste se podría concluir acerca de si es la emoción que provoca la música en humanos o son las características físicas de la onda lo que modulan la empatía. Entonces podría ser que la onda física estimulara diréctamente zonas cerebrales y no fuese el contenido emocional de la música.

9.1. Conclusiones sobre RNN-LSTM

El algoritmo DAE fue implementado exitosamente, logrando pérdidas en test por debajo de 0.1 en la versión 2, aunque fuera de lo esperado, en la versión 1 no se lograron buenos resultados debido probablemente a un problema en la implementación. El éxito en la ver-

sión 2 demuestra que se logró utilizar adecuadamente RNN-LSTM y que la estrategia de regularización fue adecuada.

La caracterización de las emociones no se logró adecuadamente, los valores de pérdidas de test no fueron buenos. Esto pudo haber sido causado por el problema de la no idéntica distribución en el conjunto de test y el conjunto de entrenamiento o por una implementación o diseño defectuoso.

Existe un detalle importante a incluir, que es que la música de la base de datos en muchos casos tenía letra, lo que podría implicar que las emociones que despertaron en los anotadores se debían parcialmente al contenido de lenguaje y por ende se tiene la siguiente conclusión. Usar solamente las características físicas del sonido no incluiría el contenido lingüístico para realizar las predicciones de las emociones.

9.2. Conclusiones sobre algoritmos de Entropía

Todos los repertorios elegidos son de mas o menos la misma época (finales de Siglo XVII y siglo XVIII). La relación entre intervalos ascendentes y descendentes, muestran que la música de esa época tiende a tener más ascendencias, para todos los autores, tal vez sea una característica de la música natural en general, pero habría que aumentar la base de datos, e incluir repertorios de compositores de otras épocas, para ver concluir adecuadamente respecto a si es la música en general o la música de esa época la que tiende a tener más intervalos ascendentes. Por otro lado, la fecha de nacimiento de los compositores está relacionada con los promedios de entropías normalizadas y de entropías de primer orden, siendo mayores para los repertorios de compositores que nacieron en el siglo XVIII con respecto a los que nacieron en el siglo XVII.

Sobre el enfoque de usar entropía para categorizar la música se puede decir que es adecuado, ya que los diferentes repertorios, a pesar de ser de períodos similares y por ende tener una estructura armónica similar, en este trabajo se ha visto que es posible capturar características musicales de cada repertorio. En particular CSD condicional diferencia los repertorios de buena manera. Sería ideal buscar más posibles parámetros y combinaciones de estos para ver cuales definen mejor los estilos de cada repertorio. Por otro lado, como trabajo a futuro sería bueno extrapolar a una mayor variedad de compositores, de preferencia de épocas más diferentes. Además, utilizando herramientas más sofisticadas, podría obtenerse medidas de información directamente de ondas de audio, de archivos mp3, wav, etc. Esto junto con Aprendizaje de Máquinas podría entregar resultados insospechados sobre cómo se categoriza la música según diferentes parámetros, como estilos, emoción que produce, como afecta al cerebro, etc.

Se puede agregar que cada compositor tiene cierta tendencia a maximizar la entropía en alguna característica en particular. Bach Chorales mostró la mayor entropía normalizada para intervalos melódicos, El Clavecín Bien Temperado mostró la mayor entropía normalizada para Contour y la menor entropía normalizada para otros parámetros. En contraste, Barbershop Quartets mostró la menor entropía normalizada para intervalos melódicos. Los cuartetos de

cuerdas mostraron las mayores entropías normalizadas para Texture.

Sería ideal poder ver la complejidad por cada canción, pero la herramienta de la Entropía no sería adecuada para ello, puesto que se requiere de una muestra mayor para que sea representativa.

Esta parte ha sido lograda con éxito y se corroboran los resultados del estudio [79].

9.3. Trabajo futuro

Como trabajo futuro se propone modificar diferentes características del algoritmo supervisado de ML, como la estructura de entrenamiento, las estrategias de regularización y la forma de elegir los hiper-parámetros. Se propone implementar un algoritmo que ejecute DAE e inmediatamente ejecute el aprendizaje supervisado, para así poder mejorar los hiperparámetros de ambos modelos con respecto a la pérdida en la validación final. Además se propone usar *Random Search* en vez de *Grid Search* e implementar un algoritmo de optimización que utilice los gradientes [91, 77].

Además, sería de gran valor poder implementar en conjunto con lo expuesto, un algoritmo que tomara como entrada las letras de las canciones. O dándole otro enfoque, podrían difuminarse las voces al momento de que se generaran las anotaciones, para que así los anotadores no tuviesen un estímulo lingüístico.

Con respecto a la Teoría de la Información se recomienda buscar o generar un repertorio que contenga música simple para así poder contrastar de mejor manera la complejidad. Además, se propone diseñar una forma para obtener la complejidad de cada canción y no de un repertorio en su totalidad, para así poder tener mejor resultados en el experimento, ya que en este el individuo no escuchará todo el repertorio.

Se propone diseñar un algoritmo que utilice ML y al mismo tiempo la Teoría de la información, sobre las mismas piezas musicales, es decir, en primer lugar, generar un algoritmo que sea capaz de obtener las complejidades de la música, basándose en la TI, a partir de la onda del sonido. Y en segundo lugar, idear la manera de fusionar estos indicadores.

Es de interés mencionar que el diseño del experimento y de las herramientas que utiliza, tiene como objetivo observar como afectan las características de la música en la respuesta empática instantánea, pero más allá que eso, sería de gran valor investigar cómo afecta la música en el aprendizaje de la respuesta empática de las personas, pudiendo ser muy útil para evaluar la importancia de la música en el sistema educativo establecido, ya que, si el acercamiento a la música generase una mejora en las respuestas empáticas de las personas, entonces sería fundamental para construir una sociedad más justa.

Bibliografía

[1] Russell J. A. A circumplex model of affect. *Journal of Personality and Social Psychology 39:1161-1178*, 1980. 2, 41

[2] Priyanka A. Abhang, Bharti W. Gawali, and Suresh C. Mehrotra. Chapter 2 - technological basics of eeg recording and operation of apparatus. In Priyanka A. Abhang, Bharti W. Gawali, and Suresh C. Mehrotra, editors, *Introduction to EEG- and Speech-Based Emotion Recognition*, pages 19 – 50. Academic Press, 2016. ix, 86

[3] Anna Alajanki, Yi-Hsuan Yang, and Mohammad Soleymani. Emotion in music task at mediaeval 2015. *PLOS ONE*, 2016. 54, 55

[4] Anna Aljanaki, Mohammad Soleymani, and Yi-Hsuan Yang. Mediaeval 2015 - emotion in music: Task overview. `https://es.slideshare.net/multimediaeval/mediaeval-2015-emotion-in-music-task-overview`. Accessed: 2019-05-24. viii, 53

[5] Anna Aljanaki, Yi-Hsuan Yang, and Mohammad Soleymani. Developing a benchmark for emotional analysis of music. *PLOS ONE*, 12(3):1–22, 03 2017. viii, 53

[6] Matías Alvarado and Francisco Meneses-Bautista. Pronóstico del tipo de cambio usd/mxn con redes neuronales de retropropagación. *Research in Computing Science*, 113:97 – 110, 05 2017. vii, 25

[7] A. Avenanti, I. Minio-Paluello, I. Bufalari, and Aglioti S.M. The pain of a model in the personality of an onlooker: influence of state-reactivity and personality traits on embodied empathy for pain. *Neuroimage 44, 275-283.*, 2009. 89

[8] A. D. Baird, I. E. Scheffer, and S. J. Wilson. Mirror neuron system involvement in empathy: A critical look at the evidence. *Social Neuroscience, 6(4), 327-335.*, 2011. 83

[9] Gino Baltazar. Cpu vs gpu in machine learning. `https://www.datascience.com/blog/cpu-gpu-machine-learning`. Accessed: 2019-05-24. viii, 47

[10] S. Baron-Cohen. How to build a baby that can read minds: Cognitive mechanisms in mindreading. *Cahiers de Psychologie Cognitive/ Current Psychology of Cognition, 13, 513-552.*, 1994. 1, 83

[11] C.D. Batson, J. Fultz, and P. A. Schoenrade. Distress and empathy: two qualitatively distinct vicarious emotions with different motivational consequences. *Journal of*

Personality., 55(1):19–39, 1987. 83

[12] Joseph Beeney, Robert Franklin, Kenneth Levy, and Reginald B Adams. I feel your pain: Emotional closeness modulates neural responses to empathically experienced rejection. *Social neuroscience*, 6:369–76, 03 2011. 84

[13] A.J. Bell and T.J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation, 7, 1129–59.*, 1995. 92

[14] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. *IEEE Tr. Neural Nets.*, page 1183–1195, 1993. 26

[15] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *In IEEE International Conference on Neural Networks, San Francisco. IEEE Press (invited paper).*, page 1183–1195, 1994. 26

[16] D. E. Berlyne. Conflict, arousal, and curiosity. *New York: McGraw-Hill*, 1960. viii, 44

[17] D. E. Berlyne. Aesthetics and psychobiology. *New York: Appleton-Century-Crofts*, 1971. 44

[18] B. C. Bernhardt and T. Singer. The neural basis of empathy. *Annual Review of Neuroscience. 35, 1-23.*, 2012. 1, 83

[19] J. Block. The q-sort method in personality assessment and psychiatric research. *Springfield, IL: Charles C. Thomas.*, 1961. 87

[20] Tomas Borovicka, Marcel Jirina Jr., Pavel Kordik, and Marcel Jirina. Selecting representative data sets. In Adem Karahoca, editor, *Advances in Data Mining Knowledge Discovery and Applications*, chapter 2. IntechOpen, Rijeka, 2012. vii, 38

[21] B. K Bryant. The interpersonal context of success: Differing empathy in children and adults. in n. eisenberg, & j. strayer (eds.). *Empathy and its development (pp. 361-373). Cambridge, UK: Cambridge University Press.*, 1987. 83

[22] B. Calvo-Merino, D.E. Glaser, J. Grèzes, R.E. Passingham, and P. Haggard. Action Observation and Acquired Motor Skills: An fMRI Study with Expert Dancers. *Cerebral Cortex*, 15(8):1243–1249, 12 2004. 83

[23] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997. 38

[24] Laura K. Case, Jaime Pineda, and Vilayanur S. Ramachandran. Common coding and dynamic interactions between observed, imagined, and experienced motor and somatosensory activity. *Neuropsychologia*, 79:233 – 245, 2015. Special Issue: Sensory Motor Integration. 83

[25] O. Chapelle, B. Scholkopf, and A. Zien, Eds. Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, March 2009. 39

[26] Yawei Cheng, Po-Lei Lee, Chia-Yen Yang, Ching-Po Lin, Daisy Hung, and Jean Decety. Gender differences in the mu rhythm of the human mirror-neuron system. *PloS one*, 3:e2113, 02 2008. 84

[27] L. Christov-Moore and M. Iacoboni. Self-other resonance, its control and prosocial inclinations: brain-behavior relationships. *Hum. Brain Mapp. 37, 1544-1558.*, 2016. 89

[28] J. E. Cohen. Information theory and music. *Behavioral Science 7(2):137-163*, 1962. 43

[29] Marcello Costantini, Gaspare Galati, Antonio Ferretti, Massimo Caulo, Armando Tartaro, Gian Luca Romani, and Salvatore Maria Aglioti. Neural Systems Underlying Observation of Humanly Impossible Movements: An fMRI Study. *Cerebral Cortex*, 15(11):1761–1767, 02 2005. 83

[30] Marcello Costantini, Gaspare Galati, Gian-Luca Romani, and Salvatore Aglioti. Empathic neural reactivity to noxious stimuli delivered to body parts and non-corporeal objects. *The European journal of neuroscience*, 28:1222–30, 10 2008. 84

[31] Karen D. Davis, Stephen J. Taylor, Adrian P. Crawley, Michael L. Wood, and David Mikulis. Functional mri of pain- and attention-related activations in the human cingulate cortex. *Journal of neurophysiology*, 77:3370–80, 07 1997. 84

[32] M. Davis. A multidimensional aproach to individual differences in empathy. *JSAS Catalog of selected documents in Psychology 10,85*, 1980. 86, 89

[33] M. Davis. Measuring individual differences in empathy: Evidence for a multidimensional aproach. *Journal of Personality and Social Psychology Vol. 44 No. 1 113-126*, 1983. 83

[34] M. H. Davis. Empathy: A social psychological approach. *Madison, W1: Westview*, 1996. 89

[35] S. Debener, M. Ullsperger, M. Siegel, K. Fiehler, D.Y. von Cramon, and A.K. Engel. Trial-by-trial coupling of concurrent electroencephalogram and functional magnetic resonance imaging identifies the dynamics of performance monitoring. *Journal of Neuroscience, 25, 11730–7.*, 2005. 92

[36] Jean Decety and S.D. Hodges. A social cognitive neuroscience model of human empathy. *Bridging Social Psychology: Benefits of Transdisciplinary Approaches*, pages 103–109, 01 2007. 84

[37] Bonifacio Del Rio and Carlos Serrano. Fundamentos de las redes neuronales artificiales: hardware y software. *Scire: Representación y organización del conocimiento. Vol. 1, Nº 1.*, pages 103–125, 1995. vii, 20

[38] S W Derbyshire. Exploring the pain "neuromatrix". *Current review of pain*, 4(6):467–77, 2000. 84

[39] Niklas. Donges. Machine learning project from a to z. `https://machinelearning-blog.com/2017/11/19/fsgdhfju/`. Accessed: 2010-04-10. vii, 14

[40] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pages 2121 – 2159, 2011. 33

[41] Justin Durham, Chanda Rooney, and Robert D. Mather. Mirror neuron systems and eeg mu wave desynchronization while observing pain. *no publicado*, 12 2016. 84

[42] T. Eichele, K. Specht, M. Moosmann, M. L. Jongsma, R. Q. Quiroga, H. Nordby, and K. Hugdahl. Assessing the spatiotemporal evolution of neuronal activation with single-trial event-related potentials and functional mri. *Proceedings of the National Academy of Sciences of the United States of America, 102, 17798–803.*, 2005. 92

[43] T. Evgeniou and M. Pontil. Support vector machines: Theory and applications. *Conference Paper.*, 2001. 17

[44] Y. Fan and S. Han. Temporal dynamic of neural mechanisms involved in empathy for pain: An event-related brain potential study. *Neuropsychologia, 46(1), 160.173*, 2008. 83, 90

[45] Simona Farina, Michele Tinazzi, Domenica Le Pera, and Massimiliano Valeriani. Pain-related modulation of the human motor cortex. *Neurological research*, 25(2):130–42, Mar 2003. 84

[46] J. Fiser and R. N. Aslin. Statistical learning of new visual feature combinations by infants. *Proceedings of the National Academy of Sciences 99(24):15822-15826*, 2002. 44

[47] H. Gastaut. Etude electrocorticographique de la reactivite des rythmes rolandiques. *Rev Neurol (Paris)*, 87:176–182, 1952. 84

[48] V. Gazzola, L. Aziz-Zadeh, and C. Keysers. Empathy and the somatotopic auditory mirror system in humans. *Curr. Biol. 16, 1824-1829*, 2006. 89

[49] B. Orr Genevieve. Momentum and learning rate adaptation. `https://www.willamette.edu/~gorr/classes/cs449/momrate.html`. Accessed: 2010-04-10. vii, 33

[50] Luciano Giromini, Piero Porcelli, Donald Viglione, Laura Parolin, and Jaime A Pineda. The feeling of movement: Eeg evidence for mirroring activity during the observations of static, ambiguous stimuli in the rorschach cards. *Biological psychology*, 85:233–41, 10 2010. 84

[51] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. vii, 10, 11, 13, 21, 26, 27, 28, 29

[52] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. Understanding the affect of developers: Theoretical background and guidelines for psychoempirical software engineering. *7th Intl. Workshop on Social Software Engineering.*, pages 25–32, 07 2015. vii, 42

[53] S. J. Guastello. Physiological synchronization in a vigilance dual task. *Nonlinear Dynamics, Psychology, and Life Science (NDPLS), 20(1), 49-80.*, 2016. 83

[54] S. J. Guastello, D. E. Marra, C. Perna, J. Castro, M. Gomez, and A. F. Peressini. Physiological synchronization in emergency response teams: Subjective workload, drivers, and empaths. *Dynamics, Psychology, and Life Science (NDPLS), 20(2), 223-270.*, 2016. 83

[55] Shihui Han, Yan Fan, Xiaojing Xu, Jungang Qin, Bing Wu, Xiaoying Wang, Salvatore M Aglioti, and Lihua Mao. Empathic neural responses to others' pain are modulated by emotional contexts. *Human brain mapping*, 30(10):3227–37, Oct 2009. 84

[56] P. Harris, C. Johnson, D. Hutton, G. Andrews, and T. Cooke. Young children's theory of mind and emotion. *Cognition and Emotion, 3, 379-400.*, 1989. 1, 83

[57] L. Hiller and R. Fuller. Structure and information in webern's symphonie, op. 21. *Journal of Music Theory 11(1):60-115*, 1967. 43, 44

[58] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma thesis*, pages 18, 401, 403, 1991. 26

[59] D. Huron. The humdrum toolkit. *Available online at music-cog.shio-state.edu/Humdrum/index.html*, 1993. 62

[60] D. Huron. Sweet anticipation: Music and the psychology of expectation. *Cambridge, Massachusetts: MIT Press*, 2006. 43

[61] Marco Iacoboni. Imitation, empathy, and mirror neurons. *Annual Review of Psychology*, 60(1):653–670, 2009. PMID: 18793090. 83, 84

[62] P. Jackson and A. N. Meltzoff. How do we perceive the pain of others? a window into the neural processes involved in empathy. *Neuroimage 24, 771-779*, 2005. 89

[63] Navdeep Jaitly and Geoffrey E. Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. *International Conference on Machine Learning (ICML)*, 2013. 39

[64] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Accessed: 2010-04-10. vii, 15

[65] Ai Kawakami, Kiyoshi Furukawa, and Kazuo Okanoya. Music evokes vicarious emotions in listeners. *Frontiers in Psychology*, 5:431, 2014. 86

[66] JiYoung Kim and SeongYoel Kim. The effects of visual stimuli on eeg mu rhythms in healthy adults. *Journal of Physical Therapy Science*, 28(6):1748–1752, 2016. 84

[67] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego*, 2015. 34

[68] L. Knopoff and W. Hutchinson. Information theory for musical continua. *Journal of Music Theory 25(1):17-44*, 1981. 43, 44

[69] L. Knopoff and W. Hutchinson. Entropy as a measure of style: The influence of sample length. *Journal of Music Theory 27(1):75-97*, 1983. 43, 44

[70] D. Krahenbuehl and J. E. Coons. Information as a measure of experience in music. *Journal of Aesthetics and Art Criticism 17(4):510-522*, 1959. 43

[71] Christof Kuhbandner, Reinhard Pekrun, and Markus A Maier. The role of positive and negative affect in the "mirroring" of other persons' actions. *Cognition and Emotion*, 24(7):1182–1190, 2010. 85

[72] Claus Lamm, Jean Decety, and Tania Singer. Meta-analytic evidence for common and distinct neural networks associated with directly experienced pain and empathy for pain. *NeuroImage*, 54(3):2492 – 2502, 2011. 84

[73] F. Lerdahl. Cognitive constraints on compositional systems. *In J. Sloboda, ed. Generative Processes in Music: The Psychology of Performance, Improvisation, and Composition. Oxford: Oxford University Press, pp. 231-259*, 1988. 44

[74] X. Li, X. Meng, H. Li, J. Yang, and J. Yuan. The impact of mood on empathy for pain: Evidence from an eeg study. *Psychophysiology, 54, 1311-1322.*, 2017. 2, 86

[75] T. Lipps. Empathy, inner imitation and sense-feelings. *In M. Rader (Ed.), A modern book of esthetics (1979l 5th ed., pp. 374-382). New York, NY: Holt, Rinehart, and Winston.*, 1903. 83

[76] Robert M. Stern, William Ray, and Karen Quigley. *Brain: Electroencephalography and Imaging*, pages 79–105. 12 2000. 84, 87

[77] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2113–2122. JMLR.org, 2015. 96

[78] Louis-Alexandre Marcoux, Pierre-Emmanuel Michon, Julien I A Voisin, Sophie Lemelin, Etienne Vachon-Presseau, and Philip L Jackson. The modulation of somatosensory resonance by psychopathic traits and empathy. *Frontiers in human neuroscience*, 7:274, 2013. 84

[79] E. H. Margulis and A. P. Beatty. Musical style, psychoaesthetics, and prospects for entropy as an analytic tool. *The MIT Press*, 2008. vi, 43, 45, 77, 96, 130, 131

[80] Franziska Martin, Vera Flasbeck, Elliot C. Brown, and Martin Brüne. Altered murhythm suppression in borderline personality disorder. *Brain Research*, 1659:64 – 70, 2017. 85

[81] Albert Mehrabian and Norman Epstein. A measure of emotional empathy. *Journal of*

Personality, 40(4):525–543, 1972. 83, 87

[82] L. Meyer. Style and music: Theory, history, and ideology. *Phildaelphia: University of Pennsylvania Press*, 1957. 43

[83] Sayantanava Mitra, S Haque Nizamie, Nishant Goyal, and Sai Krishna Tikka. Mu-wave activity in schizophrenia: Evidence of a dysfunctional mirror neuron system from an indian study. *Indian journal of psychological medicine*, 36(3):276–81, Jul 2014. 85

[84] Sayantanava Mitra, S. Haque Nizamie, Nishant Goyal, and Sai Krishna Tikka. Event related desynchronisation of mu-wave over right sensorimotor cortex at baseline may predict subsequent response to antipsychotics in schizophrenia. *Asian Journal of Psychiatry*, 14:19 – 21, 2015. 85

[85] Roy Mukamel, Arne D. Ekstrom, Jonas Kaplan, Marco Iacoboni, and Itzhak Fried. Single-neuron responses in humans during execution and observation of actions. *Current Biology*, 20(8):750–756, Apr 2010. 83

[86] Joaquín Navarro. *El mundo de la música*. Oceano, Barcelona (España), 1ra edition, 1998. 1

[87] N. Nettelheim. A bibliography of statistical applications in musicology. *Musicology Australia 20:94-106*, 1997. 43

[88] Lindsay M Oberman, Edward M Hubbard, Joseph P McCleery, Eric L Altschuler, Vilayanur S Ramachandran, and Jaime A Pineda. Eeg evidence for mirror neuron dysfunction in autism spectrum disorders. *Brain research. Cognitive brain research*, 24(2):190–8, Jul 2005. 84, 93

[89] Lindsay M. Oberman, Jaime A. Pineda, and Vilayanur S. Ramachandran. The human mirror neuron system: A link between action observation and social skills. *Social Cognitive and Affective Neuroscience*, 2(1):62–66, 03 2007. 83

[90] Ogoshi Y Momose S Takezawa T Ogoshi, S and Y. Mitsuhashi. Mu rhythm suppression during the imagination of observed action. *Conf Proc IEEE Eng Med Biol Soc. 2013:4310-3*, 2013. 84

[91] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 737–746. JMLR.org, 2016. 96

[92] J. H. Pfeifer, M. Iacoboni, J. C. Mazziota, and M. Dapretto. Mirroring others' emotions relates to empathy and interpersonal competence in children. *Neuroimage 29, 2076-2085.*, 2008. 89

[93] Gert Pfurtscheller and Colin Andrew. Event-related changes of band power and coherence: Methodology and interpretation. *Journal of clinical neurophysiology : official publication of the American Electroencephalographic Society*, 16:512–9, 12 1999. 84

[94] J. A. Pineda, M. Grichanik, V. Williams, M. Trieu, H. Chang, and C. Keysers. Eeg sensorimotor correlates of translating sounds into actions. *HFrontiers in Neuroscience, 7(203), 1-9.*, 2013. 93

[95] Jaime A. Pineda. The functional significance of mu rhythms: Translating "seeing" and "hearing" into "doing". *Brain Research Reviews*, 50(1):57 – 68, 2005. 84

[96] Jaime A. Pineda, Luciano Giromini, Piero Porcelli, Laura Parolin, and Donald Viglione. Mu suppression and human movement responses to the rorschach test. *Neuroreport*, 22:223–6, 02 2011. 84

[97] R. C. Pinkerton. Information theory and melody. *Scientific American 194(2):77-86.*, 1956. 43, 44

[98] P Rainville, G H Duncan, D D Price, B Carrier, and M C Bushnell. Pain affect encoded in human anterior cingulate but not somatosensory cortex. *Science (New York, N.Y.)*, 277(5328):968–71, Aug 1997. 84

[99] Pierre Rainville. Brain mechanisms of pain affect and pain modulation. *Current opinion in neurobiology*, 12(2):195–204, Apr 2002. 84

[100] Petra Ritter, Matthias Moosmann, and Arno Villringer. Rolandic alpha and beta eeg rhythms' strengths are inversely related to fmri-bold signal in primary somatosensory and motor cortex. *Human brain mapping*, 30:1168–87, 04 2009. 84

[101] G Rizzolatti and L. Craighero. The mirror neuron system. *Annual Review of Neuroscience 27:169-192*, 27:169–192, 2004. 83

[102] H. Robinds and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics, vol. 22, pp. 400–407*, 1951. 32

[103] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXviv*, 2017. 32

[104] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. vii, 38

[105] Oliver Sacks. *Musicofilia - Relatos de la Música y el cerebro.* Anagrama, España, 3. ed edition, 2007. 1

[106] J. Saffran. Statistical language learning: Mechanisms and constraints. *Current Directions in Psychological Science 12(4):110-113*, 2003. 44

[107] J. Saffran and R. N. Aslin. Statistical learning by 8-month-old infants. *Science 274:1926-1928*, 1996. 44

[108] J. et al. Saffran. Statistical learning of tone sequences by human infants and adults. *Cognition 70(1):27-52*, 1999. 44

[109] R. San Martın, L.G. Appelbaum, S.A. Huettel, and M.G. Woldorff. Cortical brain activity reflecting attentional biasing toward reward-predicting cues covaries with economic decision-making performance. *Cerebral Cortex, 26, 1–11.*, 2016. 92

[110] S. Sanei and J. Chambers. Eeg signal processing. *Hoboken, New Jersey: John Wiley and Sons Ltd*, 2007. 87

[111] Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization.,. *arXiv preprint*, 2013. 34

[112] C. Scheibe, M. Ullsperger, W. Sommer, and H.R. Heekeren. Effects of parametrical and trial-to-trial variation in prior prob-ability processing revealed by simultaneous electroencephalo-gram/functional magnetic resonance imaging. *Journal of Neuroscience, 30, 16709–17.*, 2010. 92

[113] Beate Seibt, Andreas Mühlberger, Katja Likowski, and Peter Weyers. Facial mimicry in its social setting. *Frontiers in Psychology*, 6:1122, 2015. 85

[114] C. Shannon. A mathematical theory of communication. *Reprinted with corrections from The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656*, 1948. 4

[115] J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, pages 67–79, 1991. 39

[116] T. Singer, B. Seymour, J. O'Doherty, H. Kaube, R.J. Dolan, and C.D. Frith. Empathy for pain involves the affective but not sensory components of pain. *Science*, 303(5661):1157–1162, 2004. 84

[117] Tania Singer and Claus Lamm. The social neuroscience of empathy. *Annals of the New York Academy of Sciences*, 1156, 03 2009. 84

[118] F. Singh, J. Pineda, and K. S. Cadenhead. Eeg sensorimotor correlates of translating sounds into actions. *Association of impaired EEG mu wave suppression, negative symptoms and social functioning in biological motion processing in first episode of psychosis 130(1-3), 182-186.*, 2011. 93

[119] J. L. Snyder. Entropy as a measure of musical style: The influence of a priori assumptions. *Music Theory Spectrum 12(1):121-160*, 1990. 43

[120] Marianne Sonnby-Borgström, Peter Jönsson, and Owe Svensson. Emotional empathy as related to mimicry reactions at different levels of information processing. *Journal of Nonverbal Behavior*, 27(1):3–23, Mar 2003. 84

[121] Marianne Sonnby-Borgström. Automatic mimicry reactions as related to differences in emotional empathy. *Scandinavian journal of psychology*, 43:433–43, 01 2003. 84

[122] Michael J. Spilka, Christopher J. Steele, and Virginia B. Penhune. Gesture imitation in musicians and non-musicians. *Experimental Brain Research*, 204(4):549–558, Aug 2010. 84

[123] McKinnon C. M. Mar R. A. Spreng, R. N. and B. Levine. The toronto empathy questionnaire: Scale development and initial validation of a factor-analytic solution to multiple empathy measures. *Journal of Personality Assessment, 91(1), 62-71.*, 2009. 83

[124] staff (2014). Medical gallery of blausen medical 2014. `https://blausen.com`. Accessed: 2010-05-28. ix, 85

[125] Y. Tang and C. Eliasmith. Deep networks for robust visual recognition. *Proceedings of the 27th International Conference on Machine Learning, June 21-24, 2010, Haifa, Israel.*, 2010. 39

[126] Mohsen Tavakol and Reg Dennick. Making sense of cronbach's alpha. *International journal of medical education*, 2:53–55, Jun 2011. 28029643[pmid]. 16

[127] OpenCV Dev Team. Introduction to support vector machines. `http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`. Accessed: 2010-04-10. vii, 17

[128] D. Temperley. Music and probability. *Cambridge, Massachusetts: MIT Press*, 2007. 43

[129] Patrizia Thoma, Christine Friedmann, and Boris Suchan. Empathy and social problem solving in alcohol dependence, mood disorders and selected personality disorders. *Neuroscience and biobehavioral reviews*, 37(3):448–70, Mar 2013. 84

[130] E Vachon-Presseau, M Roy, M O Martel, G Albouy, J Chen, L Budell, M J Sullivan, P L Jackson, and P Rainville. Neural processing of sensory and emotional-communicative information associated with the perception of vicarious pain. *NeuroImage*, 63(1):54–62, Oct 2012. 84

[131] Kandice J. Varcin, Phoebe E. Bailey, and Julie D. Henry. Empathic deficits in schizophrenia: The potential role of rapid facial mimicry. *Journal of the International Neuropsychological Society*, 16(4):621–629, 2010. 84

[132] Ruysschaert L. Wiersema J. R. Handl A. Pattyn G. Roeyers H. Warreyn, P. Infants' mu suppression during the observation of real and mimicked goal-directed actions. *Dev Sci. 2013 Mar;16(2):173-185*, 2016. 84

[133] Wikipedia. Cuda. `https://es.wikipedia.org/wiki/CUDA`. Accessed: 2019-05-24. viii, 48

[134] W. M. Wundt. Principles of physiological psychology. *Leipzig: Engelmann*, 1874. 44

[135] J. E. Youngblood. Style as information. *Journal of Music Theory 2(1):24-35*, 1958. 43, 44

Apéndice A

Códigos

A.1. Obtención de Entropía

```
import os
import math
import numpy as np
from os import listdir
import commands

class kernfile:
        def __init__(self,folder,file):
                self.file=file
                self.folder=folder
                self.address=folder+file
                self.archivo=open(self.address,'r')
                self.entropia=0

        def eliminarSubspines(self):
                fields = commands.getstatusoutput('fields -s '+self.address)
                os.system('cp '+self.address+' '+self.folder+'final'+self.file
                    )
                while True:
                        if len(fields[1].split('\n'))>2:
                                os.system('extractx -e '+self.folder+'final'+
                                    self.file+' > '+self.folder+'AUX'+self.file
                                    )
                                os.system('mv '+self.folder+'AUX'+self.file+' 
                                    '+self.folder+'final'+self.file)
                                fields = commands.getstatusoutput('fields -s '
                                    +self.folder+'final'+self.file)
                        elif len(fields[1].split('\n'))==2:
                                return kernfile(self.folder,'final'+self.file)
                                break
                        elif len(fields[1].split('\n'))==1:
                                print 'archivo '+self.file+' fallado'
                                break

        def separar(self,folder2):
```

```
        spines= int(commands.getstatusoutput('extractx -C '+self.
            folder+self.file)[1])
        for i in range(spines):
                os.system('extract -f '+str(i+1)+' '+self.folder+self.
                    file+' > '+self.folder+folder2+str(i+1)+self.file)

def filtrar(self):
        os.system("humsed '/=/s/[0-9]*//g;s/=:|!/=/g;s/\./=/g;s/=-/=/g
            ;s/==/=/g;s/=||/=/g;s/r/=/g;s/p/=/g;s/f/=/g' "+self.folder+
            self.file+" > "+self.folder+"onlyequalsign_"+self.file)
        return kernfile(self.folder,'onlyequalsign_'+self.file)

def estirar(self):
        archivo=open(self.address,'r')
        lineas=archivo.readlines()
        archivo.close()
        i=0
        nuevaslineas=[]
        for line in lineas:
                if line[0] in ['-','1','2','3','4','5','6','7','8','9'
                    ,'0']:
                        if len(line.split(' '))>1:
                                nuevalinea=''
                                acorde=line.split(' ')
                                for nota in acorde:
                                        nuevalinea=nuevalinea+nota+'\n
                                            '
                                nuevaslineas.append([i,nuevalinea])
                i=i+1
        nuevaslineas.reverse()
        for linea in nuevaslineas:
                lineas[linea[0]]=linea[1]
        archivo=open(self.address,'w')
        archivo.writelines(lineas)
        archivo.close()

def semit(self):
        os.system('semits -tx '+self.folder+self.file+' > '+self.
            folder+'semit_'+self.file)
        return kernfile(self.folder,'semit_'+self.file)

def duplicar(self):
        os.system('context -o = -n 2 '+self.address+' > '+self.folder+
            'doble.krn')
        return kernfile(self.folder,'doble.krn')

def condicional(self):
        H=0
        info=commands.getstatusoutput('infot -p -x = '+self.address)
        probacum={}
        prob={}
        lineas=info[1].split('\n')
        for linea in lineas:
                prob[linea.split('\t')[0]]=float(linea.split('\t')[1])
                if linea.split(' ')[0] in probacum.keys():
                        probacum[linea.split(' ')[0]]+=float(linea.
```

```
                                        split('\t')[1])
                        else:
                                probacum[linea.split('␣')[0]]=float(linea.
                                    split('\t')[1])
                for linea in lineas:
                        H=H-prob[linea.split('\t')[0]]*np.log2(prob[linea.
                            split('\t')[0]]/probacum[linea.split('␣')[0]])
                return H

        def key(self):
                k= commands.getstatusoutput('key␣'+self.address)
                return k[1].split('␣')[2]

        def trans(self,k):
                os.system('transpose␣-k␣'+k+'␣'+self.address+'␣>␣'+self.folder
                    +'AUXRCCSD'+self.file)
                return kernfile(self.folder,'AUXRCCSD'+self.file)

        def kern(self):
                os.system('kern␣-x␣'+self.address+'␣>␣'+self.folder+'kern'+
                    self.file)
                return kernfile(self.folder,'kern'+self.file)

        def pc(self):
                os.system('pc␣'+self.address+'␣>␣'+self.folder+'pc_'+self.file
                    )
                return kernfile(self.folder,'pc_'+self.file)
        def xdelta(self):
                os.system("xdelta␣-b␣=␣"+self.address+"␣|␣␣humsed␣'/\[/d'␣>␣"+
                    self.folder+"xdeltaDirect_"+self.file)
                return kernfile(self.folder,'xdeltaDirect_'+self.file)
        def xdeltaabs(self):
                os.system("xdelta␣-a␣-b␣=␣"+self.address+"␣|␣␣humsed␣'/\[/d'␣>
                    ␣"+self.folder+"xdeltaUndirect_"+self.file)
                return kernfile(self.folder,'xdeltaUndirect_'+self.file)

        def reassign(self):
                os.system('recode␣-i␣**Xsemits␣-f␣'+self.folder+'reassign␣'+
                    self.address+'␣>␣'+self.folder+'reassign_'+self.file)
                return kernfile(self.folder,'reassign_'+self.file)

class varioskernel:
        def __init__(self,folder):
                self.kerns=listdir(folder)
                self.folder=folder

        def unir(self):
                file=''
                for x in self.kerns:
                        file=file+'␣'+self.folder+x
                os.system('humcat␣-s'+file+'␣>␣'+self.folder+'final.krn')
                return kernfile(self.folder,'final.krn')

def main(fold):
        folder=fold
        files=listdir(folder)
```

```
        os.system('mkdir '+folder+'Separados')
        for file in files:
                a=kernfile(folder,file)
                sinsubspines=None
                try :
                        sinsubspines=a.eliminarSubspines()
                except:
                        pass
                if sinsubspines!=None:
                        sinsubspines.semit().filtrar().separar('Separados/')
        for x in listdir(folder+'Separados'):
                a=kernfile(folder+'Separados/',x)
                a.estirar()
        varios=varioskernel(fold+'Separados/')
        print fold
        infor=commands.getstatusoutput('infot -x = -s '+varios.unir().address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
        Hnorm=H/Hmax
        print 'entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

        files2=listdir(folder)
        for file in files2:
                if file in files or file =='Separados':
                        pass
                else:
                        os.system('rm '+folder+file)
        files3=listdir(folder+'Separados/')
        for file in files3:
                if file=='final.krn' or file=='pc_final.krn':
                        pass
                else:
                        os.system('rm '+folder+'Separados/'+file)

def maincond(address):
        cond=kernfile(address+'Separados/','final.krn')
        print 'Entropia condicional para '+address+': '+str(cond.duplicar().
            filtrar().condicional())

def mainRCCSD(fold):
        folder=fold
        files=listdir(folder)
        os.system('mkdir '+folder+'RCCSD')
        for file in files:
                if file=='Separados':
                        pass
                else:
                        a=kernfile(folder,file)
                        sinsubspines=None
                        try :
                                sinsubspines=a.eliminarSubspines()
                        except:
                                pass
                        if sinsubspines!=None:
                                sinsubspines.kern().trans(sinsubspines.key()).
```

```
                                        semit().filtrar().separar('RCCSD/')
        for x in listdir(folder+'RCCSD'):
                a=kernfile(folder+'RCCSD/',x)
                a.estirar()
        varios=varioskernel(fold+'RCCSD/')

        print 'RCCSD: '+fold
        infor=commands.getstatusoutput('infot -x = -s '+varios.unir().address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
        Hnorm=H/Hmax
        print 'entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

        files2=listdir(folder)
        for file in files2:
                if file in files or file =='Separados' or file == 'RCCSD':
                        pass
                else:
                        os.system('rm '+folder+file)
        files3=listdir(folder+'RCCSD/')
        for file in files3:
                if file=='final.krn':
                        pass
                else:
                        os.system('rm '+folder+'RCCSD/'+file)

def maincondRCCSD(address):
        cond=kernfile(address+'RCCSD/','final.krn')
        print 'RCCSD'
        print 'Entropia condicional para '+address+': '+str(cond.duplicar().
            filtrar().condicional())

def mainCSD(fold):
        a=kernfile(fold+'RCCSD/','final.krn')
        pca=a.pc()
        infor=commands.getstatusoutput('infot -x = -s '+pca.address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
        Hnorm=H/Hmax
        print 'CSD'+fold
        print 'entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

def maincondCSD(fold):
        cond=kernfile(fold+'RCCSD/','pc_final.krn')
        print 'CSD'
        print 'Entropia condicional para '+fold+': '+str(cond.duplicar().
            filtrar().condicional())

def mainTexture(fold):
        cant={}
        H=0
        total=0
        for x in listdir(fold):
                if x=='Separados' or x == 'RCCSD':
```

```
                        pass
                else:
                        info=commands.getstatusoutput('vox '+fold+x+' | infot
                            -n -x =')
                        for i in info[1].split('\n'):
                                try:
                                        if i.split('\t')[0] in cant.keys():
                                                cant[i.split('\t')[0]]+=int(i.
                                                    split('\t')[1])
                                        else:
                                                cant[i.split('\t')[0]]=int(i.
                                                    split('\t')[1])
                                except:
                                        pass
        for n in cant:
                total=total+cant[n]
        for n in cant:
                p=float(cant[n])/total
                H=H-p*np.log2(p)
        pequi=1/float(len(cant))
        Hmax=-np.log2(pequi)
        Hnorm=H/Hmax
        print 'Texture, '+fold+': H= '+str(H)
        print '         Hnorm= '+str(Hnorm)

def maincondTexture(fold):
        lista=listdir(fold)
        os.system('mkdir '+fold+'VOX')
        for x in lista:
                if x=='Separados' or x == 'RCCSD':
                        pass
                else:
                        os.system('vox '+fold+x+' > '+fold+'VOX/'+x)
        a=varioskernel(fold+'VOX/')
        print 'Texture, '+fold+': Entropia condicional= '+str(a.unir().
            duplicar().filtrar().condicional())

def mainDMI(fold):
        a=kernfile(fold+'Separados/','final.krn')
        infor=commands.getstatusoutput('infot -s -x =' +a.xdelta().address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
        Hnorm=H/Hmax
        print fold+', DMI: entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

def mainconDMI(fold):
        a=kernfile(fold+'Separados/','onlyequalsign_doble.krn')
        print 'DMI, '+fold+': Entropia condicional= '+str(a.xdelta().
            condicional())

def mainUMI(fold):
        a=kernfile(fold+'Separados/','final.krn')
        infor=commands.getstatusoutput('infot -s -x =' +a.xdeltaabs().address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
```

```
        Hnorm=H/Hmax
        print fold+', UMI: entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

def mainconUMI(fold):
        a=kernfile(fold+'Separados/','onlyequalsign_doble.krn')
        print 'UMI, '+fold+': Entropia condicional= '+str(a.xdeltaabs().
            condicional())

def mainContour(fold):
        a=kernfile(fold+'/Separados/','xdeltaDirect_final.krn')
        infor=commands.getstatusoutput('infot -s -x= '+a.reassign().address)
        H=float(infor[1].split('\n')[4].split('\t')[1])
        Hmax=float(infor[1].split('\n')[3].split('\t')[1])
        Hnorm=H/Hmax
        print fold+', Contour: entropia de primer orden: '+str(H)
        print 'entropia normalizada: '+str(Hnorm)

def mainconContour(fold):
        a=kernfile(fold+'Separados/','xdeltaDirect_onlyequalsign_doble.krn')
        print 'Contour, '+fold+': Entropia condicional= '+str(a.reassign().
            condicional())

if __name__=='__main__':
        carpetas=next(os.walk('.'))[1]
        for p in carpetas:
                main(p+'/')
        for p in carpetas:
                maincond(p+'/')
        for p in carpetas:
                mainRCCSD(p+'/')
        for p in carpetas:
                maincondRCCSD(p+'/')
        for p in carpetas:
                mainCSD(p+'/')
        for p in carpetas:
                maincondCSD(p+'/')
        for p in carpetas:
                mainTexture(p+'/')
        for p in carpetas:
                maincondTexture(p+'/')
        for p in carpetas:
                mainDMI(p+'/')
        for p in carpetas:
                mainconDMI(p+'/')
        for p in carpetas:
                mainContour(p+'/')
        for p in carpetas:
                mainconContour(p+'/')
```

A.2. Obtención de emociones

A continuación se muestra el código donde están todas las funciones necesarias para DAE y supervisado

```
import pandas as pd
import csv
from scipy.stats.stats import pearsonr
from os import listdir, mkdir
import numpy as np
import pickle
import math
import torch
import time
import datetime
import torch.nn.utils.rnn as rnn
import torch.nn as nn
import torch.multiprocessing as mp
import random as rd
from torch.utils.data import Dataset, DataLoader
import copy
import torch.multiprocessing as mp
from torch.autograd import Variable

p_sql = 'data/sqls/data3.sqlite3'
# p_sql = 'data/sqls/data.sqlite3'
# p_sql = 'data/sqls2/data.sqlite3'
f_data = 'data/diccionarios4/'
f_features = 'data/features/'
p_static = 'data/DEAM_Annotations/annotations/annotations averaged per song/
    song_level/static_annotations_averaged_songs_1_2000.csv'
# f_data = 'data/diccionarios/'
# f_daes='data/modelos_dae/'
# f_daes = 'data/modelos_dae2/'
f_daes = 'data/mod_dae_colab/'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
f_modelos_dae = 'data/modelos_dae4/'
f_infos_dae = 'data/infos_daes/'
f_modelos_super = 'data/modelos_super/'
f_infos_super = 'data/infos_supers/'


def load_obj(folder, name):
    with open(folder + name + '.pkl', 'rb') as f:
        return pickle.load(f)


def save_obj(obj, folder, name):
    with open(folder + name + '.pkl', 'wb+') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)

    # (A)crear data de features alterada, con ruido y eliminando ciertos
        valores del vector:(despues de normalizar)


# features_crossval_standarized = load_obj(f_data, '
    D_features_crossval_standarized_per_fold')
# features_test_standarized = load_obj(f_data, 'D_features_test_standarized')
# features_crossval_corto_standarized = load_obj(f_data, '
    S_features_crossval_corto_standarized_per_fold')
```

```
# valence_crossval_corto_standarized = load_obj(f_data, '
    S_valence_crossval_corto_standarized_per_fold')
# arousal_crossval_corto_standarized = load_obj(f_data, '
    S_arousal_crossval_corto_standarized_per_fold')
# features_test_corto_standarized = load_obj(f_data, '
    S_features_test_corto_standarized')
# valence_test_corto_standarized = load_obj(f_data, '
    S_valence_test_corto_standarized')
# arousal_test_corto_standarized = load_obj(f_data, '
    S_arousal_test_corto_standarized')
# features_test_corto = load_obj(f_data, 'S_features_test_corto')
# valence_test_corto = load_obj(f_data, 'S_valence_test_corto')
# arousal_test_corto = load_obj(f_data, 'S_arousal_test_corto')

def agregar_ruido(song, sd):
    ruido = np.random.normal(0, sd, song.shape)
    # print(type(song),type())
    feat_noised = song.float() + torch.from_numpy(ruido).float()
    return feat_noised


def destruir_caract(ejemplo, n):
    elim = rd.sample(range(260), n)
    for dele in elim:
        ejemplo[dele] = 0
    return ejemplo


# aplica agregar ruido con media cero y desviacion standar sd en cada song del
    batch
# (agregar_ruido(song,sd))y elimina n caracteristicas en cada instante
# (destruir_caract(instant_feature,n))
# batch is a list
def corromper_batch(batch, sd, n):
    batch_feat_song_corrupted = []

    for feat_song in batch:
        batch_feat_song_corrupted.append(agregar_ruido(feat_song, sd))
        for t, instant_feat in enumerate(batch_feat_song_corrupted[-1]):
            batch_feat_song_corrupted[-1][t] = destruir_caract(
                batch_feat_song_corrupted[-1][t], n)

    return batch_feat_song_corrupted


# normalizar data
def normalizar_data(lista):
    listaNorm = []
    data = torch.cat(lista, dim=0)
    mean_lista = torch.mean(data, dim=0)
    std_lista = torch.std(data, dim=0)
    for feat_song in lista:
        listaNorm.append((feat_song - mean_lista) / std_lista)
    return listaNorm
```

```
def target_V1(y, sos):
    y1 = []
    y_target = []
    for song in y:
        song_invert = song.flip(dims=[0])
        song2_invert = torch.cat([sos.float().to('cpu'), song_invert.float()])
        y_target.append(song2_invert[:-1,:])
        y1.append(song_invert)
    return y1, y_target


# crear train data y validation data sin normalizar para cualquier batch de
    validacion
def datatrain_dae(dicti, valid_batch, separar_train_folds=False):
    list_train = []
    list_valid = []
    if separar_train_folds:
        list_train_sep = []
    # data={}
    i = 0
    for batch in dicti:
        if batch != valid_batch:
            for song in dicti[batch]:
                list_train.append(dicti[batch][song])

            if separar_train_folds:
                list_train_sep.append(list_train[i:])
                i = len(list_train)
    for song in dicti[valid_batch]:
        list_valid.append(dicti[valid_batch][song])
    if separar_train_folds:
        # debuggeo
        # print(len(list_train_sep[0]))
        # imprime 40
        return list_train_sep, list_valid
    elif not separar_train_folds:
        return list_train, list_valid


def datatrain_super(dict_feature, dict_valence, dict_arousal, valid_batch,
    separar_train_folds=False):
    list_train_feature = []
    list_valid_feature = []
    list_train_valence = []
    list_valid_valence = []
    list_train_arousal = []
    list_valid_arousal = []
    if separar_train_folds:
        list_train_feature_sep = []
        list_train_valence_sep = []
        list_train_arousal_sep = []

    i = 0
    for batch in dict_feature:
        # print(batch,dict[batch].shape)
        if batch != valid_batch:
```

```
            for song in dict_feature[batch]:
                list_train_feature.append(dict_feature[batch][song])
                list_train_valence.append(dict_valence[batch][song].unsqueeze
                    (1))
                list_train_arousal.append(dict_arousal[batch][song].unsqueeze
                    (1))
            if separar_train_folds:
                list_train_feature_sep.append(list_train_feature[i:])
                list_train_valence_sep.append(list_train_valence[i:])
                list_train_arousal_sep.append(list_train_arousal[i:])
                i = len(list_train_feature)
    for song in dict_feature[valid_batch]:
        list_valid_feature.append(dict_feature[valid_batch][song])
        list_valid_valence.append(dict_valence[valid_batch][song].unsqueeze(1)
            )
        list_valid_arousal.append(dict_arousal[valid_batch][song].unsqueeze(1)
            )
    if separar_train_folds:
        return list_train_feature_sep, list_train_valence_sep,
            list_train_arousal_sep, list_valid_feature, list_valid_valence,
            list_valid_arousal
    elif not separar_train_folds:
        return list_train_feature, list_train_valence, list_train_arousal,
            list_valid_feature, list_valid_valence, list_valid_arousal


def datatrain_test_super(dict_feature, dict_valence, dict_arousal):
    list_test_feature = []
    list_test_valence = []
    list_test_arousal = []
    for song in dict_feature:
        list_test_feature.append(dict_feature[song])
        list_test_valence.append(dict_valence[song].unsqueeze(1))
        list_test_arousal.append(dict_arousal[song].unsqueeze(1))
    return list_test_feature, list_test_valence, list_test_arousal


def datatrain_test_dae(dicti):
    list_test = []
    for song in dicti:
        list_test.append(dicti[song])
    return list_test


'''
#transformar pad en tensor plano
def aplanar(padded,lengths):
  plana=torch.Tensor([]).to(device)
  for j in range(len(padded[0])):
    plana=torch.cat([plana,padded[:lengths[j],j]])
  return plana
'''


# standarizar elementos de todos las song de cada fold a media y desviacion
    estandar de su fold
```

```
def standarize_to_fold(dicti):
    sigma = []
    dicti_standar = copy.deepcopy(dicti)
    for fold in dicti:
        tensor_fold = torch.cat([dicti[fold][song] for song in dicti[fold]],
            dim=0)
        mean_fold = torch.mean(tensor_fold, dim=0)
        std_fold = torch.std(tensor_fold, dim=0)
        # print('mean_fold={},std_fold={}'.format(mean_fold[0],std_fold[0]))
        for song in dicti[fold]:
            std_song = torch.std(dicti[fold][song], dim=0)
            mean_song = torch.mean(dicti[fold][song], dim=0)
            dicti_standar[fold][song] = dicti[fold][
                                            song] * std_fold / std_song +
                                               mean_fold - mean_song *
                                               std_fold / std_song
            # print('mean_song={}, std_song={}'.format(torch.mean(
                dicti_standar[fold][song],dim=0)[0],torch.std(dicti_standar[
                fold][song],dim=0)[0]))
    return dicti_standar


def standarize_to_all_fold(dicti, valid_fold):
    sigma = []
    dicti_standar = copy.deepcopy(dicti)

    tensor_train = torch.cat([dicti[fold][song] for fold in dicti for song in
        dicti[fold] if fold != valid_fold], dim=0)
    # print(tensor_train.shape)
    mean_train = torch.mean(tensor_train, dim=0)
    std_train = torch.std(tensor_train, dim=0)
    # print('mean_train={},std_train={}'.format(mean_train[0],std_train[0]))
    for fold in dicti:
        if fold != valid_fold:
            for song in dicti[fold]:
                std_song = torch.std(dicti[fold][song], dim=0)
                mean_song = torch.mean(dicti[fold][song], dim=0)
                dicti_standar[fold][song] = dicti[fold][
                                                song] * std_train / std_song +
                                                    mean_train - mean_song *
                                                   std_train / std_song
                # print('mean_song={}, std_song={}'.format(torch.mean(
                    dicti_standar[fold][song],dim=0)[0],torch.std(dicti_standar
                    [fold][song],dim=0)[0]))
        else:
            tensor_valid_fold = torch.cat([dicti[fold][song] for song in dicti
                [fold]], dim=0)
            mean_valid = torch.mean(tensor_valid_fold, dim=0)
            std_valid = torch.std(tensor_valid_fold, dim=0)
            for song in dicti[fold]:
                std_song = torch.std(dicti[fold][song], dim=0)
                mean_song = torch.mean(dicti[fold][song], dim=0)
                dicti_standar[fold][song] = dicti[fold][
                                                song] * std_valid / std_song +
                                                    mean_valid - mean_song *
                                                   std_valid / std_song
```

```
                # print('mean_song={}, std_song={}'.format(torch.mean(
                    dicti_standar[fold][song],dim=0)[0],torch.std(dicti_standar
                    [fold][song],dim=0)[0]))
    return dicti_standar


# para estandarizar los test (que no estan separados por folds)
def standarize(dicti):
    sigma = []
    dicti_standar = copy.deepcopy(dicti)

    tensor_test = torch.cat([dicti[song] for song in dicti], dim=0)
    mean = torch.mean(tensor_test, dim=0)
    std = torch.std(tensor_test, dim=0)
    for song in dicti:
        std_song = torch.std(dicti[song], dim=0)
        mean_song = torch.mean(dicti[song], dim=0)
        dicti_standar[song] = dicti[song] * std / std_song + mean - mean_song
            * std / std_song
    return dicti_standar


# Dataloader y dataset
def my_collate_dae(batch):
    data = [item[0] for item in batch]
    label = [item[1] for item in batch]
    return [data, label]


# Dataloader y dataset
def my_collate_super(batch):
    features = [item[0] for item in batch]
    valence = [item[1] for item in batch]
    arousal = [item[2] for item in batch]
    return [features, valence, arousal]


class featData_dae(Dataset):
    # Constructor
    def __init__(self, noiseFeatures, features):  # noiseFeatures y features
        son listas
        self.x = noiseFeatures
        self.y = features
        self.len = len(self.x)

    # Getter
    def __getitem__(self, index):
        return self.x[index], self.y[index]

    # Get length
    def __len__(self):
        return self.len


class featData_super(Dataset):
    # Constructor
```

```
    def __init__(self, list_features, list_valence, list_arousal):
        self.x = list_features
        self.y = list_valence
        self.z = list_arousal
        self.len = len(self.x)

    # Getter
    def __getitem__(self, index):
        return self.x[index], self.y[index], self.z[index]

    # Get length
    def __len__(self):
        return self.len
    # Crear packed sequence de una lista de canciones (song)


def packInput(songs):
    vectorized_seqs = songs
    seq_lengths = torch.FloatTensor([len(seq) for seq in vectorized_seqs]) #
        .cuda() #58,....1223
    seq_tensor = torch.zeros(
        (len(vectorized_seqs), seq_lengths.long().max(), len(vectorized_seqs
            [0][0]))).float()  # .to(device)

    for idx, (seq, seqlen) in enumerate(zip(vectorized_seqs, seq_lengths.long
        ())):
        seq_tensor[idx, :seqlen, :] = torch.FloatTensor(seq.float())

    # SORT YOUR TENSORS BY LENGTH!
    seq_lengths, perm_idx = seq_lengths.sort(0, descending=True)
    # print(perm_idx,seq_lengths)
    seq_tensor = seq_tensor[perm_idx]

    # utils.rnn lets you give (B,L,D) tensors where B is the batch size, L is
        the maxlength, if you use batch_first=True
    # Otherwise, give (L,B,D) tensors
    seq_tensort = seq_tensor.transpose(0, 1)  # (B,L,D) -> (L,B,D)

    # embed your sequences
    # seq_tensor = embed(seq_tensor)

    # pack them up nicely
    packed_input = rnn.pack_padded_sequence(seq_tensort, seq_lengths.cpu().
        numpy())  # .to(device)
    return packed_input


def igualar_largos(features, valence, arousal):
    features2 = copy.deepcopy(features)
    valence2 = copy.deepcopy(valence)
    arousal2 = copy.deepcopy(arousal)
    for batch in features2:
        for song in features2[batch]:
            if features2[batch][song].shape[0] > valence2[batch][song].shape
                [0]:
```

```
                features2[batch][song] = features2[batch][song][:valence2[
                    batch][song].shape[0], :]
                if features2[batch][song].shape[0] > arousal2[batch][song].
                    shape[0]:
                    features2[batch][song] = features2[batch][song][:arousal2[
                        batch][song].shape[0], :]
                else:
                    arousal2[batch][song] = arousal2[batch][song][:features2[
                        batch][song].shape[0]]
            else:
                valence2[batch][song] = valence2[batch][song][:features2[batch
                    ][song].shape[0]]
                if valence2[batch][song].shape[0] > arousal2[batch][song].
                    shape[0]:
                    valence2[batch][song] = valence2[batch][song][:arousal2[
                        batch][song].shape[0], :]
                else:
                    arousal2[batch][song] = arousal2[batch][song][:valence2[
                        batch][song].shape[0]]
    return features2, valence2, arousal2


def igualar_largos_test(features_test, valence_test, arousal_test):
    features_test2 = copy.deepcopy(features_test)
    valence_test2 = copy.deepcopy(valence_test)
    arousal_test2 = copy.deepcopy(arousal_test)
    for song in features_test2:
        if features_test2[song].shape[0] > valence_test2[song].shape[0]:
            features_test2[song] = features_test2[song][:valence_test2[song].
                shape[0], :]
            if features_test2[song].shape[0] > arousal_test2[song].shape[0]:
                features_test2[song] = features_test2[song][:arousal_test2[
                    song].shape[0], :]
            else:
                arousal_test2[song] = arousal_test2[song][:features_test2[song
                    ].shape[0]]
        else:
            valence_test2[song] = valence_test2[song][:features_test2[song].
                shape[0]]
            if valence_test2[song].shape[0] > arousal_test2[song].shape[0]:
                valence_test2[song] = valence_test2[song][:arousal_test2[song
                    ].shape[0], :]
            else:
                arousal_test2[song] = arousal_test2[song][:valence_test2[song
                    ].shape[0]]

    return features_test2, valence_test2, arousal_test2


# Clase padre de modelos
class RNN(nn.Module):
    def __init__(self, parametros):
        super(RNN, self).__init__()
        self.etapa = None
    def train(self):
        pass
```

```
    def test(self):
        pass

    def valid(self):
        pass


# (B)creacion del modelo lstm-linear Este!pba padding linear
class DAE(RNN):
    def __init__(self, parametros):
        super(DAE, self).__init__(parametros)
        self.input_size = parametros['input_size_dae']
        self.output_size = parametros['output_size_dae']
        self.hidden_size = parametros['hidden_size_dae']
        self.LR = parametros['LR_dae']
        self.sd = parametros['sd_dae']
        self.batch_size = parametros['batch_size_dae']
        self.n = parametros['n_dae']
        self.criterion = parametros['criterion_dae']
        self.num_layers = 1
        # self.momentum=parametros['momentum_dae']
        self.h, self.c = self.init_hidden()
        self.lstm = nn.LSTM(self.input_size, self.hidden_size, self.num_layers
            )
        self.linear = nn.Linear(self.hidden_size, self.output_size)
        self.optimizer = torch.optim.Adam(self.parameters(), lr=self.LR)
        self.v_DAE = parametros['v_DAE']
        if self.v_DAE == 'V1':
            self.h2, self.c2 = None, None
            self.lstm2 = nn.LSTM(self.input_size, self.hidden_size, self.
                num_layers)
            self.sos = torch.zeros([1, self.input_size], requires_grad=True).
                to(device)
            self.sosb = torch.zeros([1, self.batch_size, self.output_size]).to
                (device)
            self.linear = nn.Linear(self.hidden_size, self.output_size)
    def change_batch_size(self, new_batch_size):
        self.batch_size = new_batch_size

    def init_hidden(self):
        h = torch.zeros(self.num_layers, self.batch_size, self.hidden_size).to
            (device)
        c = torch.zeros(self.num_layers, self.batch_size, self.hidden_size).to
            (device)
        return h, c

    def train(self, list_train_sep):
        list_loss_train = []
        for fold in list_train_sep:
            dataset = featData_dae(fold, fold.copy())
            trainloader = DataLoader(dataset=dataset, batch_size=self.
                batch_size, collate_fn=my_collate_dae,
                                     drop_last=True, shuffle=True)
            for x, y in trainloader:
                x = normalizar_data(x)
```

```
                x = corromper_batch(x, self.sd, self.n)
                y = normalizar_data(y)
                if self.v_DAE == 'V2':
                    features_out = self(packInput(x).to(device))
                elif self.v_DAE == 'V1':
                    y, y_target = target_V1(y, self.sos)
                    features_out = self(packInput(x).to(device), target=
                        y_target)
                real_features = packInput(y).to(device)
                loss_train = self.criterion(features_out.data, real_features.
                    data)
                list_loss_train.append(np.sqrt(loss_train.item()))
                self.optimizer.zero_grad()
                loss_train.backward()
                self.h, self.c = self.h.detach(), self.c.detach()
                self.optimizer.step()
        return list_loss_train
    def valid(self, list_valid):
        list_loss_valid = []
        with torch.no_grad():
            dataset = featData_dae(list_valid, list_valid.copy())
            trainloader = DataLoader(dataset=dataset, batch_size=self.
                batch_size,
                                     collate_fn=my_collate_dae, drop_last=True
                                         , shuffle=True)
            for x, y in trainloader:
                x = normalizar_data(x)
                # x=corromper_batch(x,sd,n)
                y = normalizar_data(y)
                if self.v_DAE == 'V1':
                    y, y_target = target_V1(y, self.sos)
                features_out = self(packInput(x).to(device))
                real_features = packInput(y).to(device)
                loss_valid = self.criterion(features_out.data, real_features.
                    data)
                list_loss_valid.append(np.sqrt(loss_valid.item()))
        return list_loss_valid

    def test(self, features_test_standarized):
        list_test_feature = datatrain_test_dae(features_test_standarized)
        list_loss_test = []
        with torch.no_grad():
            dataset_test = featData_dae(list_test_feature, list_test_feature.
                copy())
            testloader = DataLoader(dataset=dataset_test, batch_size=self.
                batch_size,
                                    collate_fn=my_collate_dae, drop_last=True,
                                        shuffle=True)

            for features_list_input, feature_list_target in testloader:
                features_list_input = normalizar_data(features_list_input)
                feature_list_target = normalizar_data(feature_list_target)
                # lo siguiente es si drop_last=False
                # if self.batch_size != len(features_list_input):
                #     self.batch_size = len(features_list_input)
                #     self.h = self.h[:, :self.batch_size, :]
```

```
                #    self.c = self.c[:, :self.batch_size, :]
                if self.v_DAE == 'V1':
                    feature_list_target, y_target = target_V1(
                        feature_list_target, self.sos)
                features_out = self(packInput(features_list_input).to(device))
                real_features = packInput(feature_list_target).to(device)
                loss_test = self.criterion(features_out.data, real_features.
                    data)
                list_loss_test.append(np.sqrt(loss_test.item()))
                # se pondera de igual manera los loss test del batch que tiene
                    menos muestras ( el ultimo)
        return np.mean(list_loss_test),np.std(list_loss_test)

    def forward(self, x, target=None):
        output, (self.h, self.c) = self.lstm(x, (self.h, self.c))
        if self.v_DAE == 'V2':
            outpad = rnn.pad_packed_sequence(output)
            out = self.linear(outpad[0])
            out2 = rnn.pack_padded_sequence(out, outpad[1])
        elif self.v_DAE == 'V1':
            if self.etapa == 'train':
                out_lstm2, _ = self.lstm2(packInput(target).to(device), (self.
                    h, self.c))
                outpad = rnn.pad_packed_sequence(out_lstm2)
                #print(out_lstm2.data.unsqueeze(0).shape)
                #print(outpad[0].shape)
                #print(outpad[1].shape)
                #print(outpad[0].view(-1,200).view(-1,200.shape)
                #print(outpad[0].transpose(0,1).view(-1,200).shape)
                print(outpad[0].squeeze(0).shape)
                out = self.linear(outpad[0].squeeze(0).view(-1,200))
                #print(out.shape)
                #out2=type(out_lstm2)(out,out_lstm2.batch_sizes)

                #out = self.linear(outpad[0])
                #out2=rnn.pad_packed_sequence(out,out_lstm2[1])
                out2 = rnn.pack_padded_sequence(out.view(-1,5,260),outpad[1])
            elif self.etapa == 'no_train':
                largos = rnn.pad_packed_sequence(output)[1]
                out2_list = [None] * int(largos[0])
                out2_linear =[None] * int(largos[0])
                self.h2, self.c2 = copy.deepcopy(self.h), copy.deepcopy(self.c
                    )
                out2_list[0], (self.h2, self.c2) = self.lstm2(self.sosb,(self.
                    h2,self.c2))
                out2_linear[0] = self.linear(out2_list[0])
                for t in range(largos[0] - 1):
                    out2_list[t + 1], (self.h2, self.c2) = self.lstm2(
                        out2_linear[t],(self.h2,self.c2))
                    out2_linear[t+1] = self.linear(out2_list[t+1])
                out_lstm2 = torch.cat([dec for dec in out2_linear])
                out2 = rnn.pack_padded_sequence(out_lstm2, largos)
        return out2


# modelo clasificador
```

```
class LSTMsuper(RNN):
    def __init__(self, parametros, model):
        super(LSTMsuper, self).__init__(parametros)
        self.etapa = None
        self.input_size = parametros['input_size_super']
        self.output_size = parametros['output_size_super']
        self.hidden_size1 = parametros['hidden_size1_super']
        self.LR = parametros['LR_super']
        self.sd = parametros['sd_super']
        self.batch_size = parametros['batch_size_super']
        #self.batch_size_test = parametros['batch_size_test_super']
        self.n = parametros['n_super']
        self.criterion = parametros['criterion_super']
        self.num_layers = 1
        self.hidden_size2 = parametros['hidden_size2_super']
        self.hidden_size3 = parametros['hidden_size3_super']
        self.h, self.c = self.init_hidden(self.hidden_size1)
        self.h2_v, self.c2_v = self.init_hidden(self.hidden_size2)
        self.h3_v, self.c3_v = self.init_hidden(self.hidden_size3)
        self.h2_a, self.c2_a = self.init_hidden(self.hidden_size2)
        self.h3_a, self.c3_a = self.init_hidden(self.hidden_size3)
        self.dae = model
        self.lstm2valence = nn.LSTM(self.hidden_size1, self.hidden_size2)
        self.lstm3valence = nn.LSTM(self.hidden_size2, self.hidden_size3)
        self.lstm2arousal = nn.LSTM(self.hidden_size1, self.hidden_size2)
        self.lstm3arousal = nn.LSTM(self.hidden_size2, self.hidden_size3)
        self.linear_valence = nn.Linear(self.hidden_size3, self.output_size)
        self.linear_arousal = nn.Linear(self.hidden_size3, self.output_size)
        self.valence_par = list(self.lstm2valence.parameters()) + list(self.
            lstm3valence.parameters()) + list(
             self.linear_valence.parameters())
        self.arousal_par = list(self.lstm2arousal.parameters()) + list(self.
            lstm3arousal.parameters()) + list(
             self.linear_arousal.parameters())
        # self.general_param=list(self.dae.parameters())+list(self.lstm2.
            parameters())+list(self.lstm3.parameters())
        self.optimizer = torch.optim.Adam(self.dae.parameters(), lr=self.LR)
        self.optimizer_valence = torch.optim.Adam(self.valence_par, lr=self.LR
            )
        self.optimizer_arousal = torch.optim.Adam(self.arousal_par, lr=self.LR
            )
        self.v_super = parametros['v_super']

    def init_hidden(self, hid_size):
        h = torch.zeros(self.num_layers, self.batch_size, hid_size).to(device)
        c = torch.zeros(self.num_layers, self.batch_size, hid_size).to(device)

        return h, c

    def change_batch_size(self, new_batch_size):
        self.batch_size = new_batch_size

    def train(self, list_train_feature_sep, list_train_valence_sep,
        list_train_arousal_sep):
        list_loss_train = []
        list_loss_train_valence = []
```

```
list_loss_train_arousal = []
for foldindex in range(len(list_train_feature_sep)):
    # print(foldindex)
    dataset_train_lstm = featData_super(
        list_train_feature_sep[foldindex],
        list_train_valence_sep[foldindex],
        list_train_arousal_sep[foldindex])
    trainloader_lstm = DataLoader(
        dataset=dataset_train_lstm,
        batch_size=self.batch_size,
        collate_fn=my_collate_super,
        drop_last=True, shuffle=True)
    for features_list, valence_list, arousal_list in trainloader_lstm:
        features_list = normalizar_data(features_list)
        valence_list = normalizar_data(valence_list)
        arousal_list = normalizar_data(arousal_list)
        if self.v_super in ['V1','V3']:
            features_list = corromper_batch(features_list, self.sd,
                self.n)
        valence_pack = packInput(valence_list).to(device)
        arousal_pack = packInput(arousal_list).to(device)
        annotations_target = torch.cat([valence_pack.data,
            arousal_pack.data])
        valence_out, arousal_out = self(packInput(features_list).to(
            device))
        loss_train_valence = self.criterion(valence_out.data,
            valence_pack.data)
        list_loss_train_valence.append(np.sqrt(loss_train_valence.item
            ()))
        loss_train_arousal = self.criterion(arousal_out.data,
            arousal_pack.data)
        list_loss_train_arousal.append(np.sqrt(loss_train_arousal.item
            ()))
        self.optimizer_arousal.zero_grad()

        loss_train_arousal.backward(retain_graph=True)
        self.optimizer_arousal.step()
        self.optimizer_valence.zero_grad()
        loss_train_valence.backward(retain_graph=True)
        self.optimizer_valence.step()
        annotations_out = torch.cat([valence_out.data, arousal_out.
            data])
        loss_train = self.criterion(annotations_out,
            annotations_target)
        list_loss_train.append(np.sqrt(loss_train.item()))
        self.optimizer.zero_grad()
        loss_train.backward()
        self.h = self.h.detach()
        self.c = self.c.detach()
        self.h2_v = self.h2_v.detach()
        self.c2_v = self.c2_v.detach()
        self.h3_v = self.h3_v.detach()
        self.c3_v = self.c3_v.detach()
        self.h2_a = self.h2_a.detach()
        self.c2_a = self.c2_a.detach()
        self.h3_a = self.h3_a.detach()
```

```
                self.c3_a = self.c3_a.detach()
                self.optimizer.step()
        return list_loss_train, list_loss_train_valence,
            list_loss_train_arousal

    def valid(self, list_valid_feature, list_valid_valence, list_valid_arousal
        , listLoss_valid=[],
              listLoss_valid_valence=[], listLoss_valid_arousal=[]):
        with torch.no_grad():
            dataset_valid_lstm = featData_super(
                list_valid_feature,
                list_valid_valence,
                list_valid_arousal)
            trainloader_valid_lstm = DataLoader(
                dataset=dataset_valid_lstm,
                batch_size=self.batch_size,
                collate_fn=my_collate_super,
                drop_last=True, shuffle=True)

            for features_list, valence_list, arousal_list in
                trainloader_valid_lstm:
                features_list = normalizar_data(features_list)
                valence_list = normalizar_data(valence_list)
                arousal_list = normalizar_data(arousal_list)
                valence_pack = packInput(valence_list).to(device)
                arousal_pack = packInput(arousal_list).to(device)
                annotations_target = torch.cat([valence_pack.data,
                    arousal_pack.data])
                valence_out, arousal_out = self(packInput(features_list).to(
                    device))
                annotations_out = torch.cat([valence_out.data, arousal_out.
                    data])
                loss_valid = self.criterion(annotations_out,
                    annotations_target)
                loss_valid_valence = self.criterion(valence_out.data,
                    valence_pack.data)
                loss_valid_arousal = self.criterion(arousal_out.data,
                    arousal_pack.data)
                listLoss_valid.append(np.sqrt(loss_valid.item()))
                listLoss_valid_valence.append(np.sqrt(loss_valid_valence.item
                    ()))
                listLoss_valid_arousal.append(np.sqrt(loss_valid_arousal.item
                    ()))
        return listLoss_valid, listLoss_valid_valence, listLoss_valid_arousal

    def test(self, features_test_corto_standarized,
        valence_test_corto_standarized, arousal_test_corto_standarized):
        listLoss_test = []
        listLoss_test_valence = []
        listLoss_test_arousal = []
        #self.change_batch_size(self.batch_size_test)
        # self.h, self.c = self.init_hidden(self.hidden_size)
        # self.h2_v, self.c2_v = self.init_hidden(self.hidden_size2)
        # self.h3_v, self.c3_v = self.init_hidden(self.hidden_size3)
        # self.h2_a, self.c2_a = self.init_hidden(self.hidden_size2)
        # self.h3_a, self.c3_a = self.init_hidden(self.hidden_size3)
```

```
    list_test_feature, list_test_valence, list_test_arousal =
        datatrain_test_super(
         features_test_corto_standarized,
         valence_test_corto_standarized,
         arousal_test_corto_standarized)
    with torch.no_grad():
         dataset_test_lstm = featData_super(list_test_feature,
                                            list_test_valence,
                                            list_test_arousal)
         testloader_test_lstm = DataLoader(dataset=dataset_test_lstm,
                                           batch_size=self.batch_size,
                                           collate_fn=my_collate_super,
                                           drop_last=True,
                                           shuffle=True)

         for features_list, valence_list, arousal_list in
             testloader_test_lstm:
              features_list = normalizar_data(features_list)
              valence_list = normalizar_data(valence_list)
              arousal_list = normalizar_data(arousal_list)
              valence_pack = packInput(valence_list).to(device)
              arousal_pack = packInput(arousal_list).to(device)
              annotations_target = torch.cat([valence_pack.data,
                  arousal_pack.data])
              valence_out, arousal_out = self(packInput(features_list).to(
                  device))
              annotations_out = torch.cat([valence_out.data, arousal_out.
                  data])
              loss_test = self.criterion(annotations_out, annotations_target
                  )
              loss_test_valence = self.criterion(valence_out.data,
                  valence_pack.data)
              loss_test_arousal = self.criterion(arousal_out.data,
                  arousal_pack.data)
              listLoss_test.append(np.sqrt(loss_test.item()))
              listLoss_test_valence.append(np.sqrt(loss_test_valence.item())
                  )
              listLoss_test_arousal.append(np.sqrt(loss_test_arousal.item())
                  )
    return np.mean(listLoss_test),np.std(listLoss_test), np.mean(
        listLoss_test_valence),np.std(listLoss_test_valence), np.mean(
        listLoss_test_arousal), np.std(listLoss_test_arousal)

def forward(self, x):
    output, (self.h, self.c) = self.dae(x, (self.h, self.c))
    # print(output.data.shape)
    #if self.etapa == 'train':
    #    output = type(output)(output.data + torch.normal(torch.zeros_like
        (output.data), std=self.sd),
    #                          output.batch_sizes)

    output2_v, (self.h2_v, self.c2_v) = self.lstm2valence(output, (self.
        h2_v, self.c2_v))
    output3_v, (self.h3_v, self.c3_v) = self.lstm3valence(output2_v, (self
        .h3_v, self.c3_v))
```

```
output2_a, (self.h2_a, self.c2_a) = self.lstm2arousal(output, (self.
    h2_a, self.c2_a))
output3_a, (self.h3_a, self.c3_a) = self.lstm3arousal(output2_a, (self
    .h3_a, self.c3_a))
# print(output.data.shape)
outpad_a = rnn.pad_packed_sequence(output3_a)
outpad_v = rnn.pad_packed_sequence(output3_v)
out_valence = self.linear_valence(outpad_v[0])
out_arousal = self.linear_arousal(outpad_a[0])
out2_valence = rnn.pack_padded_sequence(out_valence, outpad_v[1])
out2_arousal = rnn.pack_padded_sequence(out_arousal, outpad_a[1])
return out2_valence, out2_arousal
```

Apéndice B

Otros resultados

B.1. Comparaciones de resultados de entropía con respecto a estudio previo

En esta sección se muestran gráficos y tablas que comparan el trabajo realizado en [79] con este trabajo.

	Tono	*RCCSD*	*CSD*	*Textura*	*Directed Melodic Interval*	*Undirected Melodic Interval*	*Contorno*
Bach Chorales (BC)							
Primer orden	3,68	3,97	3,05	0,13	2,96	2,13	1,45
Condicional	2,57	2,31	2,28	0,13	2,49	2,07	1,44
Normalizada	0,85	0,83	0,85	0,13	0,64	0,74	0,91
Bach Well-Tempered Clavier (BWC)							
Primer orden	4,31	4,43	3,37	1,67	3,51	2,65	1,38
Condicional	3,37	3,10	2,86	0,66	2,98	2,54	1,36
Normalizada	0,84	0,84	0,94	0,60	0,59	0,67	0,87
Barbershop Quartets (BQ)							
Primer orden	3,33	3,89	3,34	0,62	2,80	2,23	1,56
Condicional	2,53	2,40	2,44	0,43	2,60	2,13	1,53
Normalizada	0,73	0,81	0,93	0,27	0,56	0,59	0,98
Corelli Trio Sonatas (CTS)							
Primer orden	4,08	4,09	3,25	1,27	3,37	2,54	1,41
Condicional	2,97	2,79	2,62	0,76	2,85	2,44	1,39
Normalizada	0,83	0,80	0,91	0,55	0,59	0,64	0,89
Handel Trio Sonatas (HTS)							
Primer orden	4,19	4,35	3,32	1,30	3,76	2,92	1,42
Condicional	3,36	3,22	2,93	0,76	3,18	2,77	1,39
Normalizada	0,82	0,80	0,93	0,50	0,64	0,70	0,90
Telemann Violin Sonatas (TVS)							
Primer orden	4,12	4,36	3,31	0,67	3,99	3,07	1,27
Condicional	3,54	3,43	3,00	0,54	3,25	2,85	1,24
Normalizada	0,87	0,84	0,92	0,42	0,75	0,70	0,80
Haydn String Quartets (HST)							
Primer orden	4,84	4,72	3,33	2,15	3,76	2,93	1,46
Condicional	3,48	3,25	2,99	1,17	3,20	2,72	1,40
Normalizada	0,88	0,85	0,93	0,68	0,58	0,65	0,92
Mozart String Quartets (MSQ)							
Primer orden	4,48	4,53	3,23	1,94	3,70	2,86	1,42
Condicional	3,29	3,07	2,78	1,25	3,10	2,66	1,37
Normalizada	0,86	0,82	0,90	0,56	0,61	0,67	0,90

Tabla B.1: Medidas de la información obtenidas en el trabajo de Hellmuth y Beatty [79]

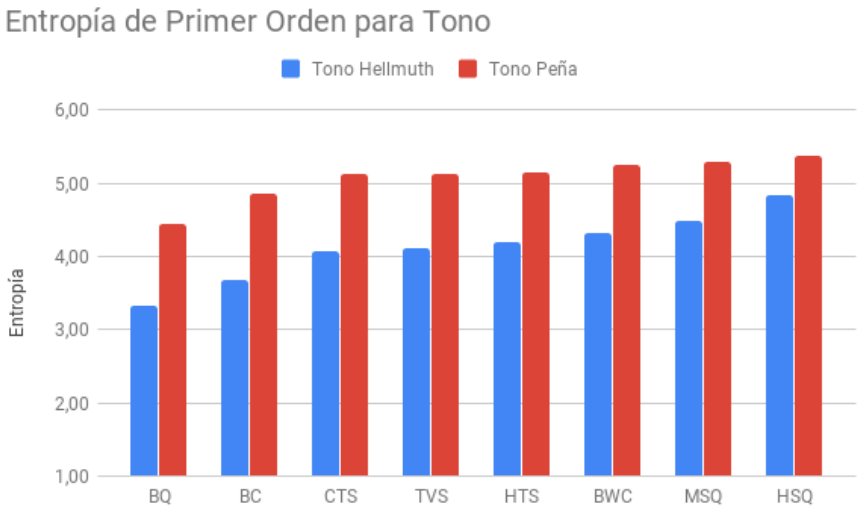

Figura B.1: Entropía de Primer Orden para Tono.

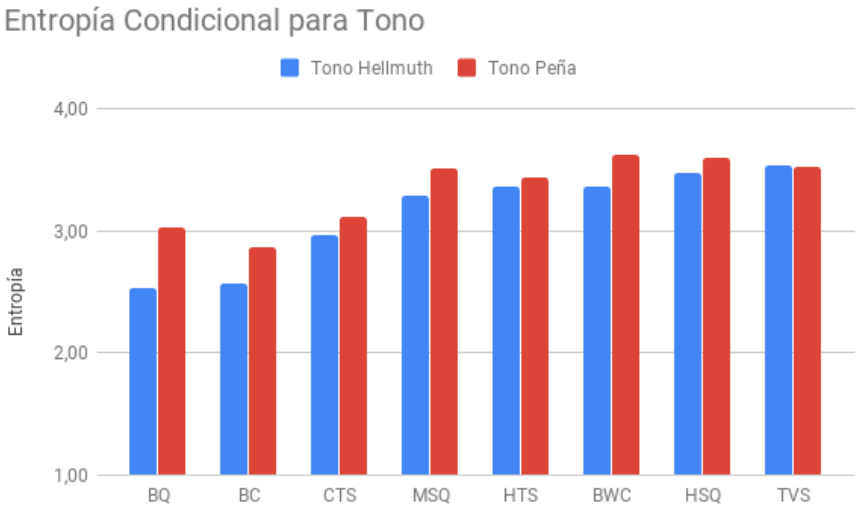

Figura B.2: Entropía Condicional para Tono.

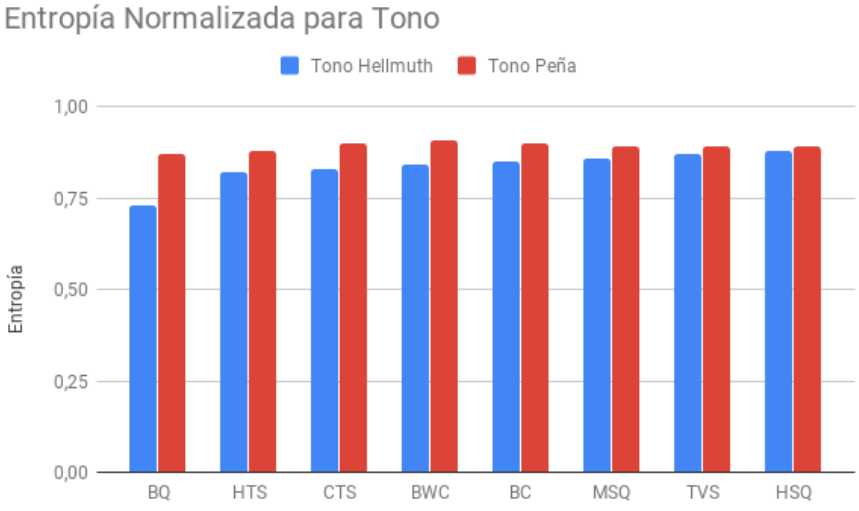

Figura B.3: Entropía Normalizada para Tono.

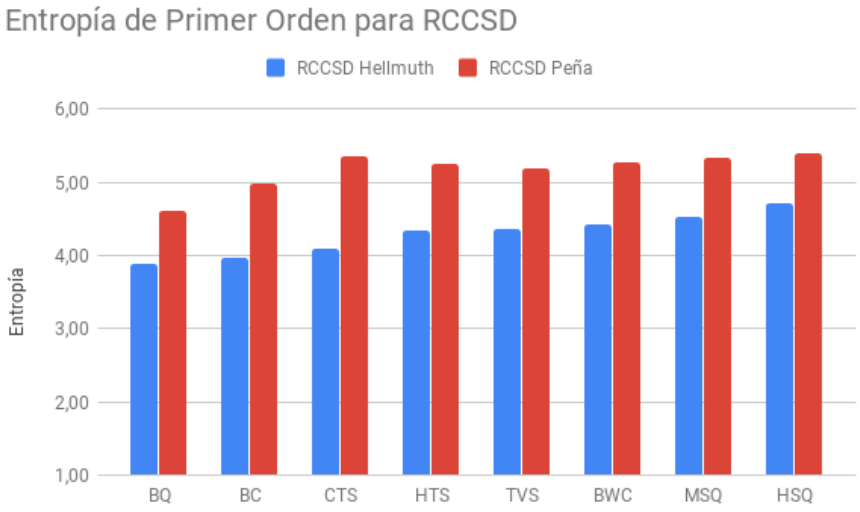

Figura B.4: Entropía de Primer Orden para RCCSD.

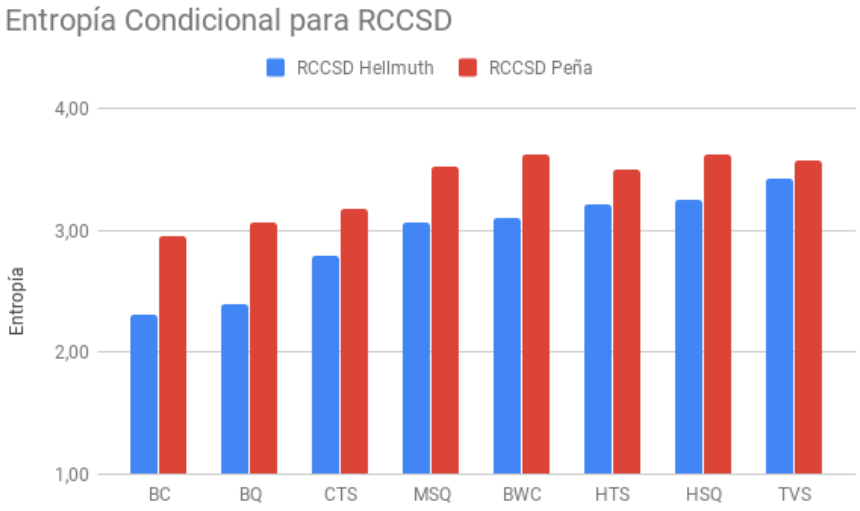

Figura B.5: Entropía Condicional para RCCSD.

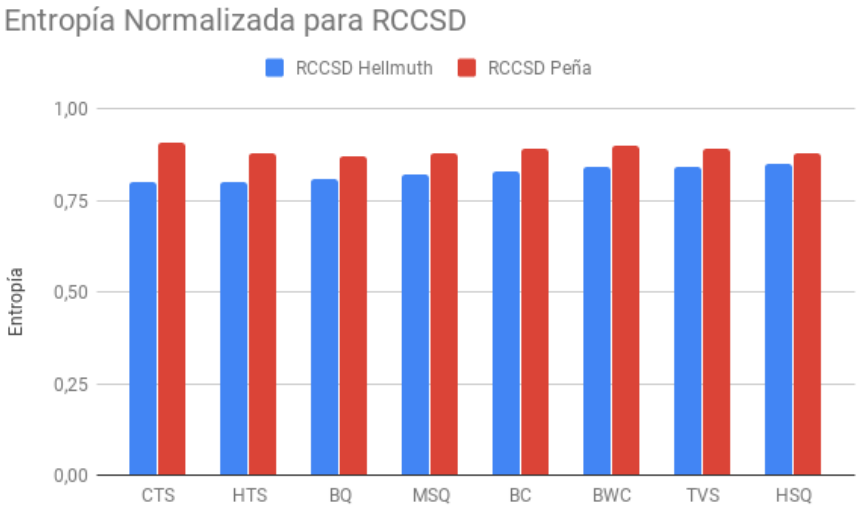

Figura B.6: Entropía Normalizada para RCCSD.

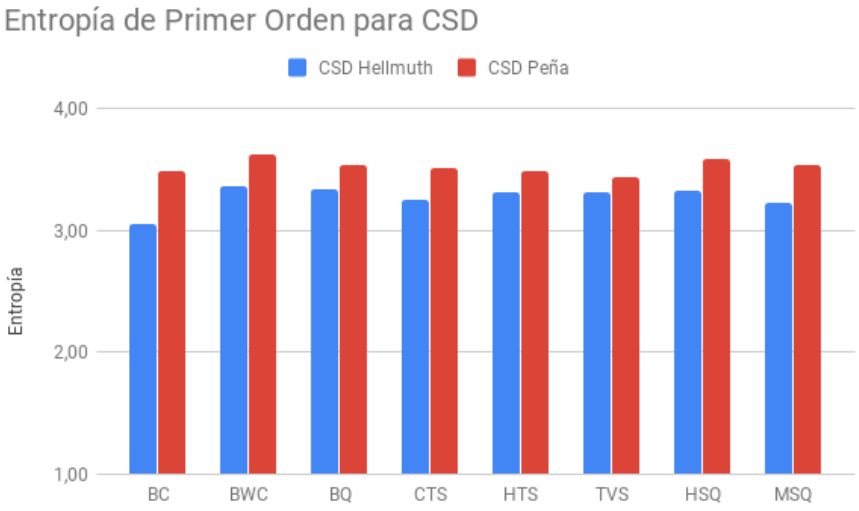

Figura B.7: Entropía de Primer Orden para CSD.

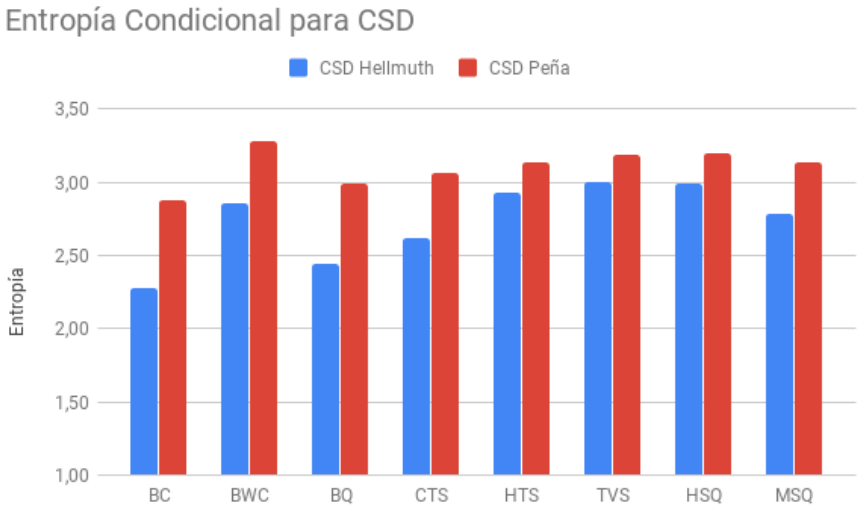

Figura B.8: Entropía Condicional para CSD.

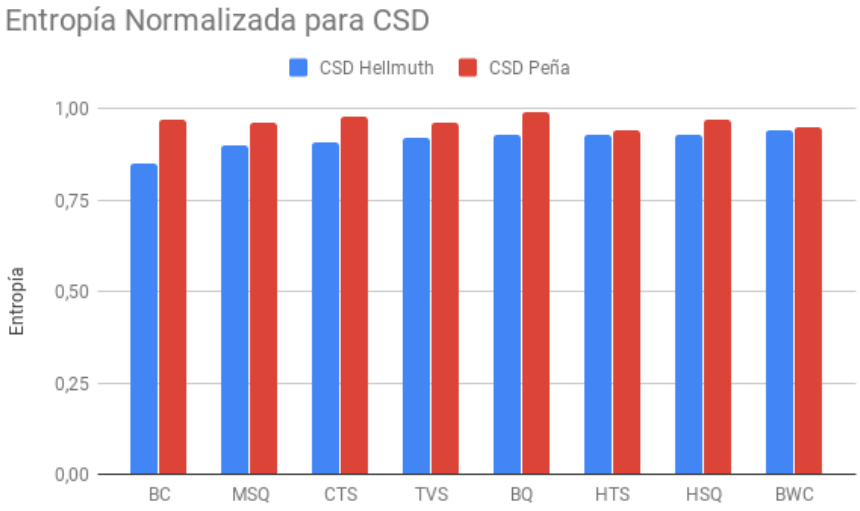

Figura B.9: Entropía Normalizada para CSD.

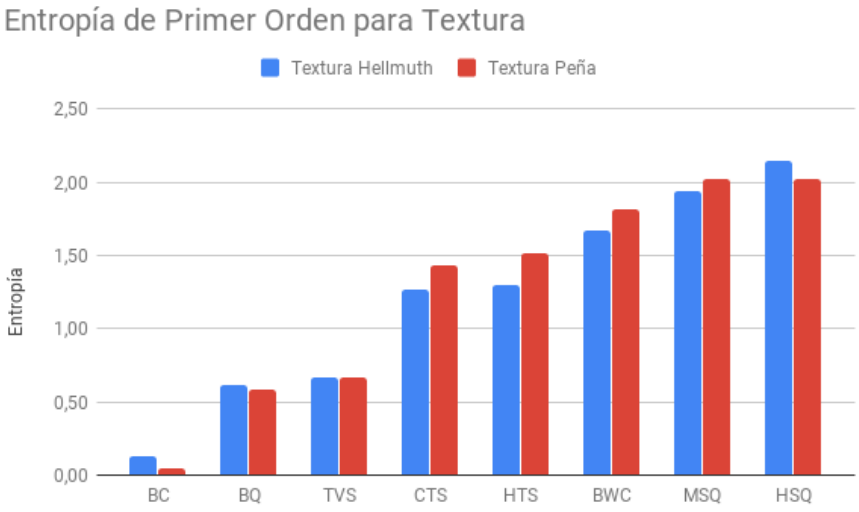

Figura B.10: Entropía de Primer Orden para Textura.

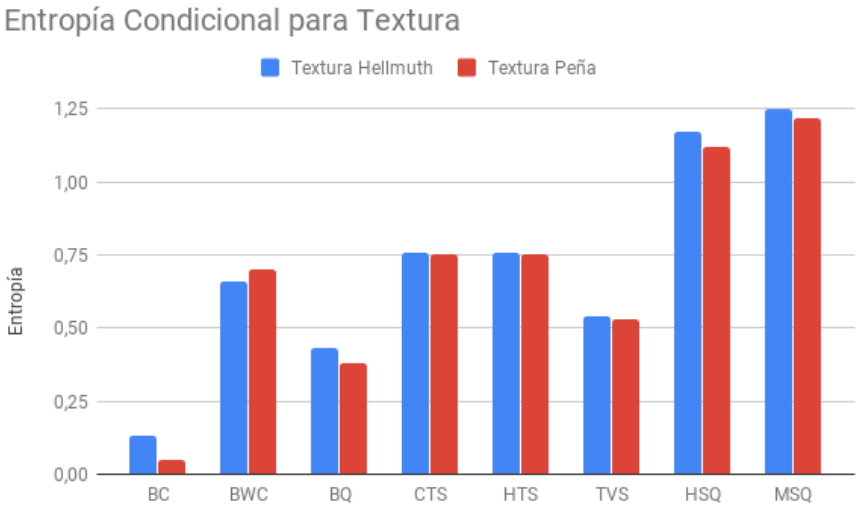

Figura B.11: Entropía condicional para Textura.

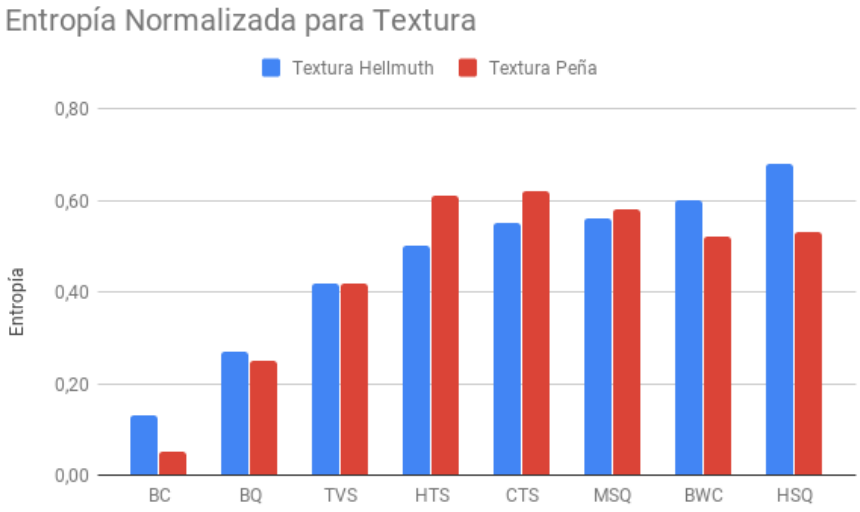

Figura B.12: Entropía Normalizada para Textura.

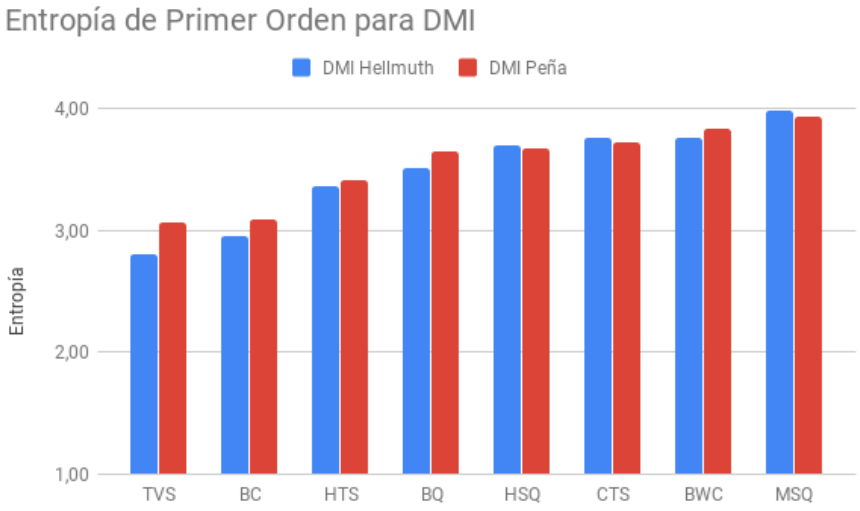

Figura B.13: Entropía de Primer Orden para DMI.

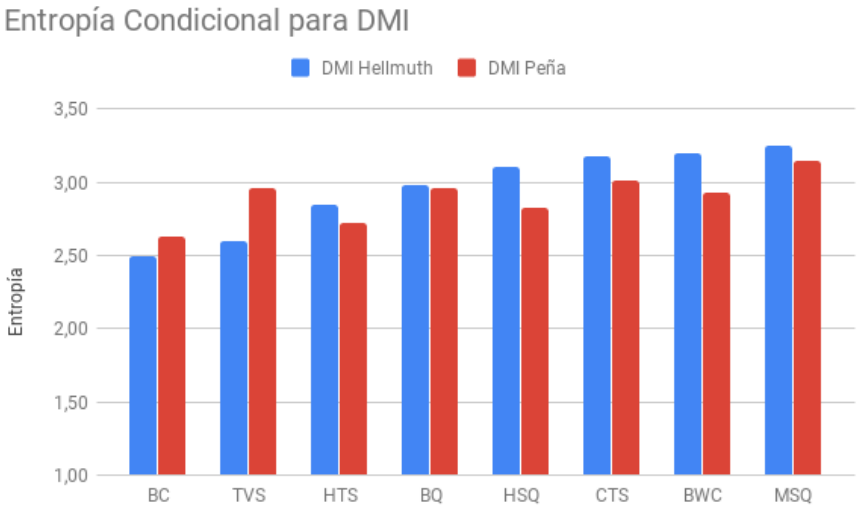

Figura B.14: Entropía Condicional para DMI.

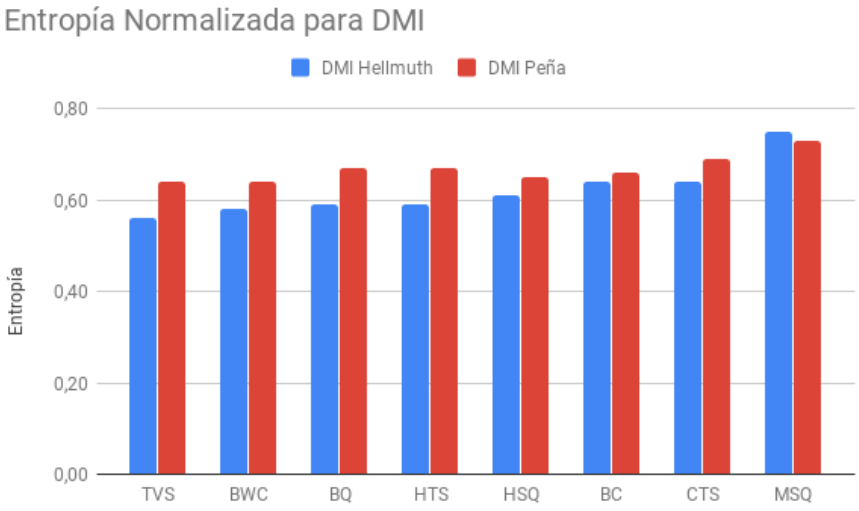

Figura B.15: Entropía Normalizada para DMI.

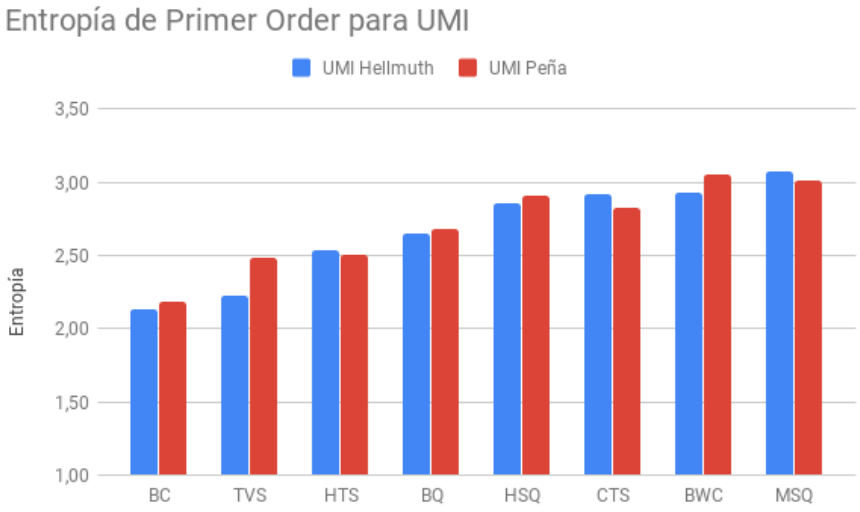

Figura B.16: Entropía de Primer Orden para UMI.

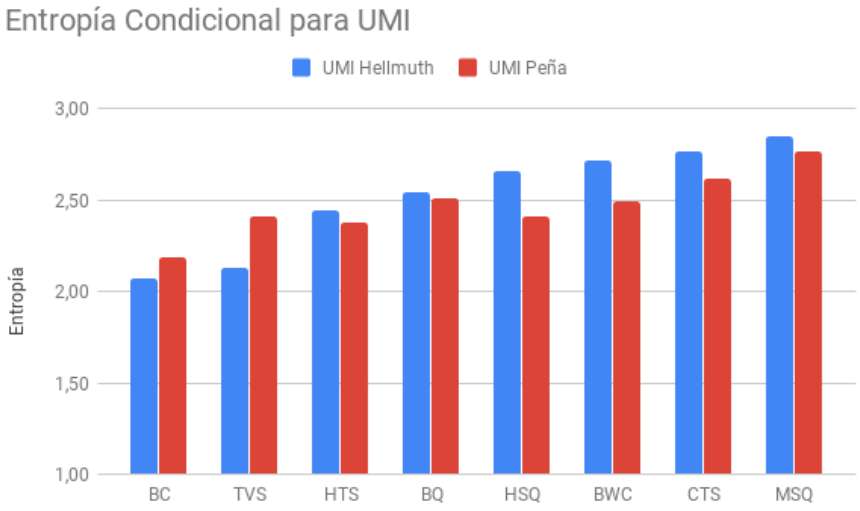

Figura B.17: Entropía Condicional para UMI.

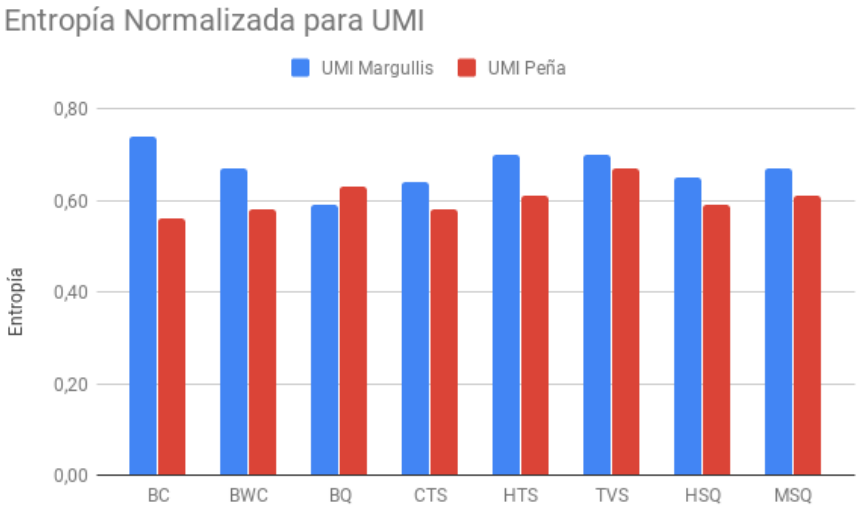

Figura B.18: Entropía Normalizada para UMI.

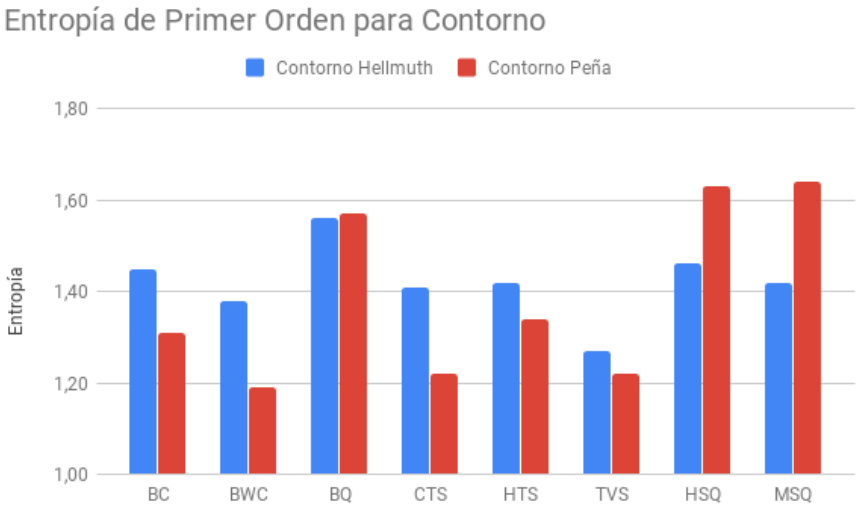

Figura B.19: Entropía de Primer Orden para Contorno.

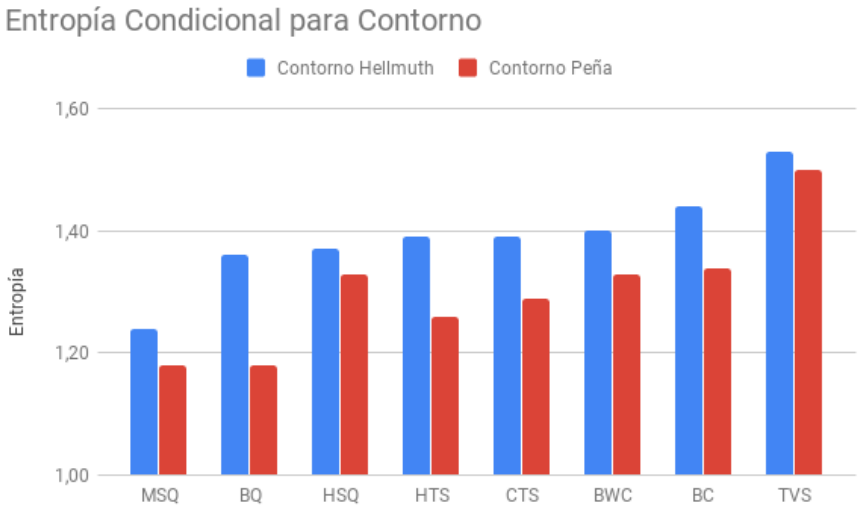

Figura B.20: Entropía Condicional para Contorno.

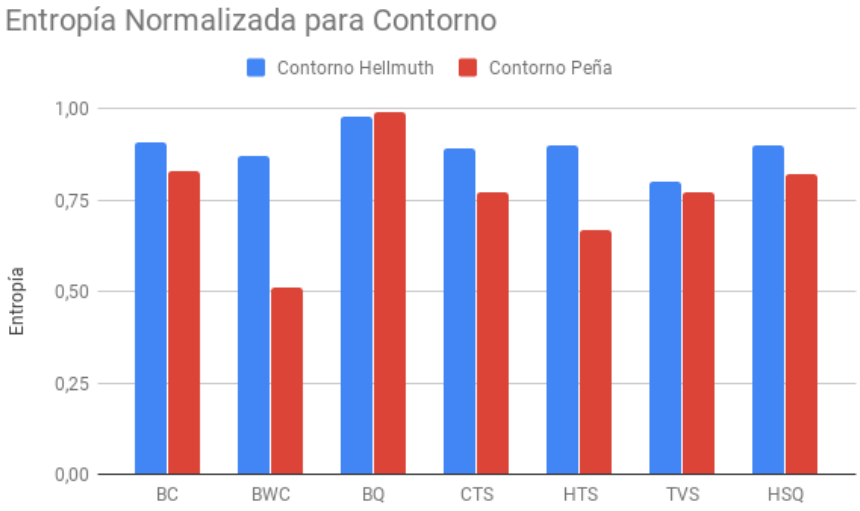

Figura B.21: Entropía Normalizada para Contorno

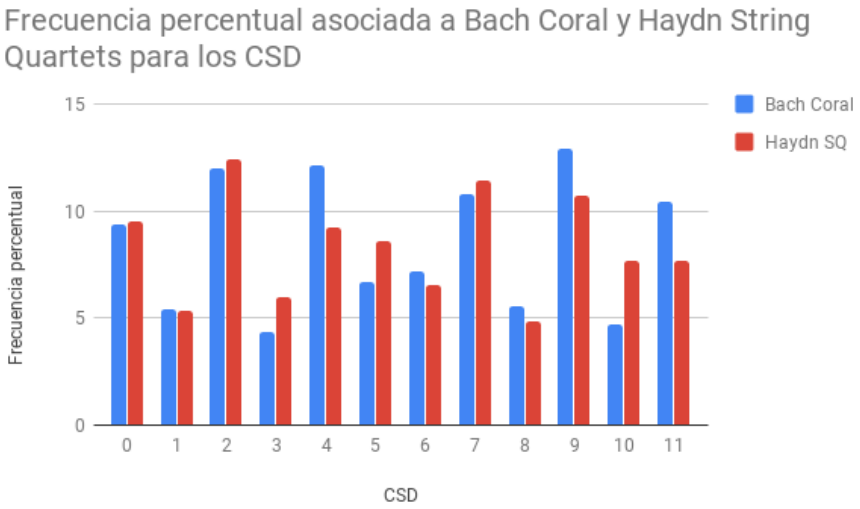

Figura B.22: Comparación entre la Frecuencia Porcentual en términos de CSD.

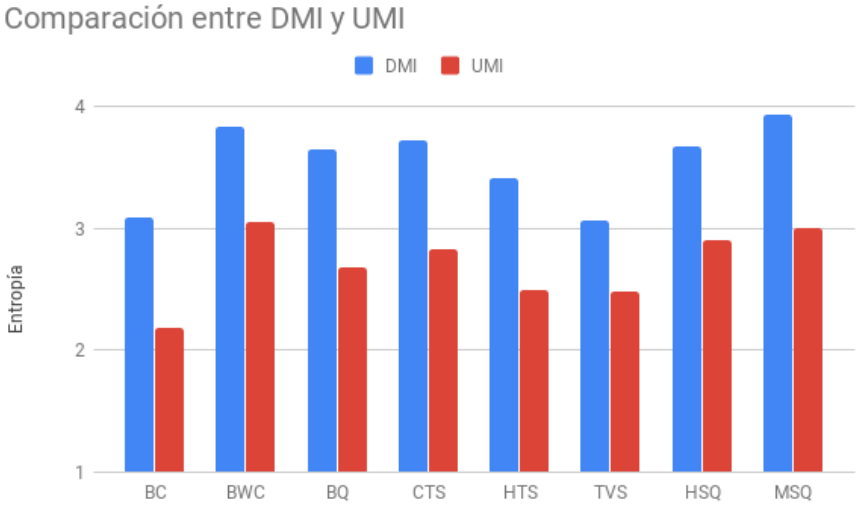

Figura B.23: Comparación entre DMI y UMI.