MelAE : A content-based next track recommendation from Mel Spectrograms

Vicente Díaz
Pontificia Universidad Católica de Chile
Santiago, Chile
vndiaz1@uc.cl

Martín Vinay
Pontificia Universidad Católica de Chile
Santiago, Chile
mlvinay@uc.cl

Abstract

In the following article we present a Mel Spectrogram to semantic embedding autoencoder model, an analysis of these embeddings, and a posterior content-based recommendation using the obtained vectors for playlist continuation, all of this based on the MPD dataset. We start by making a brief introduction of the state of the art in playlist continuation and the related concepts, after which we present the developed autoencoder model. Furthermore, we make a qualitative analysis of the obtained embeddings, to move on to a content-based next-track recommendation evaluation on a modified dataset of the MPD. We finally conclude that the developed model and semantic space embeddings are capable of representing musical style, but the applied recommendation algorithm isn't adequate for these embeddings, causing a notable error in the recommendations. We end this article leaving ideas for a promising future work.

Keywords: music, content-based, mel spectrograms, autoencoders

1 Introducción

Los servicios de *stream* se han popularizado en los últimos años, y la música no es una excepción al caso. Teniendo cada día más usuarios, las compañías de *streaming* poseen un catálogo de canciones que se cuenta en las decenas de millones de piezas. Por ejemplo, actualmente Spotify reporta tener más de 60.000.000[1] *tracks* en su catálogo, donde este número aumenta día tras día. Aunque la recomendación de contenido popular ha mostrado resultados prominentes en el último tiempo, la recomendación de contenido nuevo y poco explorado, aka, "the long tail", permanece como un desafío a ser explorado [5].

Para promover una mejora en las recomendaciones, la misma compañía de Spotify liberó un dataset conocido como Million Playlist Dataset (MPD) para el RecSys Challenge 2018, con la temática de playlist *continuation* o *next track recommendation*.

Algo notable de los resultados de este *challenge*, fue la predominancia de métodos basados en filtros colaborativos. Mientras tanto, los métodos basados en contenido, se basaban en la metadata de los archivos, no en el contenido en sí (señal de audio).

No es que sea imposible o infactible este tipo de recomendación, al contrario, según lo expuesto por Schedl en [13], el avance de las redes neuronales ha permitido un boom en el área científica de *Music Information Retrieval* (MIR), permitiendo el uso de contenido acústico (o simplemente ondas digitales) para tareas de clasificación musica [16] o generación de *embeddings* para una multiplicación de factores latentes usuario-ítem. [14][11][8].

En [14] se usa red convolucional para la generación de factores latentes de canciones tal que se obtuviese una factorización matricial. La factorización matricial se puede definir como la obtención de dos matrices, la matriz de usuarios $U \in \mathbb{R}^{u \times n}$, y la matriz de items $V \in \mathbb{R}^{n \times i}$ para u usuarios, i ítems y vector embebido de largo n) tal que la multiplicación de estas matrices, se asemeje a la matriz de interacciones implícitas $K \in \mathbb{R}^{u \times i}$.

De manera similar, en [8] se busca entrenar una red convolucional para generar *embeddings* para cada *track* para posterior similitud a un vector embebido de un usuario, y con el uso de *negative sampling* y *hinge loss*, se tiene por objetivo maximizar la similitud del vector de usuario y canciones pertenecientes a un usuario.

En [11] nuevamente se usa *matrix factorization* y se busca entrenar una red convolucional con información de espectrogramas en paralelo a capas densas con información de artistas, para generar vectores embebidos que se asemejen a los de una matriz de ítems V.

Finalmente, en [16] se busca aprender la clasificación de canciones a partir de sus espectrogramas usando únicamente nueve géneros musicales. El modelo desarrollado aprende efectivamente *embeddings* de alto nivel semántico antes de la clasificación.

Podemos ver en general, que mientras varios de los trabajos mencionados son capaces de atacar el *cold start* para nuevos ítems, la necesidad de una matriz de interacción implícita genera que las recomendaciones para nuevos usuarios sea posiblemente de baja calidad. Además, surge el problema de reducir las clasificaciones musicales a un número determinado de géneros, lo cual se podría pensar como un problema a futuro considerando la evolutiva originalidad de los artistas. Con la finalidad de superar estos dos problemas, proponemos el uso de autoencoders.

[1] https://newsroom.spotify.com/company-info/#:~:text=Discover%2C%20manage%20and%20share%20over,and%20ad%2Dfree%20listening%20experience Visitado el 16/12/20.

Los autoencoders convolucionales han demostrado en variadas investigaciones la capacidad de generar vectores de dimensionalidad reducida capaces de contener toda (o casi toda) la información de la imagen original, permitiendo usar a los autoencoders como un algoritmo de compresión de imagen [1], o recolorizar una imagen en blanco y negro [2].

Por otro lado, no necesitan información de los usuarios para su entrenamiento previo, lo que permite superar el *cold start* de estos. Finalmente, el entrenamiento de un autoencoder con la salida esperada siendo la entrada, se podría considerar como un tipo de aprendizaje no supervisado [2], lo que le da una mayor capacidad de abstracción a este tipo de redes.

2 Conceptos

2.1 Filtros colaborativos

Los filtros colaborativos, o sistemas recomendadores basados en estos, usan el historial de interacciones usuario-ítem (sea este historial explícito o implícito) para generar una base para la recomendación, usualmente usando algoritmos de *matrix factorization* y similaridades en pares *user-user*, *user-item* o *item-item*.

El principal problema de estos métodos, que se repite en variados, si es que no todos los dominios, es el *cold start*, concepto que aglomera la recomendación hacia nuevos usuarios con pocas interacciones, nuevos ítems que no han sido interactuados o una global falta de interacción usuario-ítem.

En el dominio de *next-track reccomendation*, la mayoría de los sistemas recomendadores son basados en filtros colaborativos. Esto se puede deber a una falta de datos para hacer un sistema basado en contenido o que usualmente estos entregan peores resultados a niveles de métricas como MAP y nDCG comparado a los filtros colaborativos (debido a que estos aprenden los comportamientos de los usuarios).

2.2 Filtros basados en contenido

Los filtros basados en contenido, usan un algún tipo de vector (usualmente llamados *embeddings*) que identifica a los objetos en un espacio semántico de alto nivel. Posteriormente, la recomendación es llevada a cabo generando alguna relación de similaridad entre los ítems con los que el usuario ya a interactuado, con los ítems que no han sido interactuados por este.

Estos modelos, en general no presentan *cold start* debido al espacio semántico de los *embeddings* y dependiendo del algoritmo de recomendación, pueden recomendar objetos nuevos o sin previas interacciones.

Naturalmente, los sistemas basados en contenido poseen sus problemas, como por ejemplo, la falta de poseer un set de *embeddings* de calidad para los objetos del dominio. En años recientes, con el avance de redes neuronales, se han obtenido *embeddings* para variados dominios, como es NLP (*natural language processing*) con el mayor ejemplo siendo word2vec[10], o para imágenes usando img2vec[3].

Lamentablemente, para el caso de audio o música, no existe un símil 'music2vec' o 'track2vec' que se haya popularizado, por lo que variados autores han usado como contenido para sus sistemas recomendadores la metadata de los *tracks* (eg. artista, álbum, género), en vez del contenido musical.

2.3 Autoencoders

Los autoencoders fueron originalmente introducidos como "una red neuronal con la finalidad de reconstruir su entrada"[12]. El problema formalmente definido [3], es aprender dos funciones A y B, tal que $A : \mathbb{R}^n \rightarrow \mathbb{R}^p$ y $B : \mathbb{R}^p \rightarrow \mathbb{R}^n$ tal que para una entrada x se cumpla:

$$argmin_{A,B} \quad E[\Delta(x, B(A(x)))]$$

Donde E es la expectancia sobre distribución de x, y Δ es el error de reconstrucción, comúnmente siendo la norma ℓ_2 (distancia euclidiana).

Dentro de las aplicaciones más comunes de los autoencoders, se encuentran el uso para clusterización, y su uso para reducción de dimensionalidad [4]. *Clustering* busca dividir las entradas en distintos grupos, asumiendo que para un espacio en particular (por preferencia, de una dimensionalidad considerablemente menor a la del objeto de entrada), los objetos presentan una distancia/norma baja entre miembros de un mismo grupo, y una distancia/norma considerable con objetos de otros grupos. Los autoencoders son capaces de obtener una representación con una cantidad de *features* mucho menor a la original, permitiendo el uso de algoritmos de *clustering*.

La mayor desventaja de usar autoencoder *vanilla*, es que los *embeddings* obtenidos (*features* de baja dimensionalidad) son entrenados exclusivamente para la reconstrucción y no para *clustering* o clasificación. Esto puede ser superado de varias maneras como por ejemplo, que la función de pérdida de la red sea dual entre la calidad del *clustering* (regularización) y la reconstrucción.

3 Contribuciones

En este paper presentamos:

- MelDataset: Un dataset de *Mel spectrograms* de las primeras 4500 listas del del *Million Playlist Dataset* (MPD) [6], conteniendo aproximadamente 60 mil canciones.
- Un autoencoder convolucional que permite generar vectores latentes a partir de spectrogramas que representan el contenido musical de canciones.

[2] No supervisado respecto a los *embeddings* aprendidos, ya que podría definirse el proceso como supervisado al usar la entrada como la salida esperada.

[3] No existe un *paper* original para img2vec

- Un sistema recomendador basado en contenido que permite obtener recomendaciones adecuadas para la continuación de playlists.

Puede encontrar todos los *scripts*/*notebooks* usados en este proyecto en este repositorio.

Por otro lado, puede encontrar el dataset desarrollado en el siguiente link.

Finalmente, puede encontrar el modelo entrenado en el siguiente link.

4 Modelo propuesto

Como fue mencionado previamente, nuestro objetivo es obtener vectores latentes capaces representar correctamente el contenido musical de variadas canciones. Con esto en mente, presentamos MelAE (Mel spectrograms Auto-Encoder).

4.1 MelAE

Este modelo consiste en un *autoencoder* convolucional que permite generar vectores latentes de largo definido (en este caso, 256), a partir de *Mel spectrograms*. Debido a que el dataset utilizado no presenta algún tipo de clasificación previa en las piezas/canciones, no podemos usar una rama regularizadora, por lo que se asumirá que el *embedding* obtenido a partir únicamente de la reconstrucción de espectrogramas son suficientemente significativos.

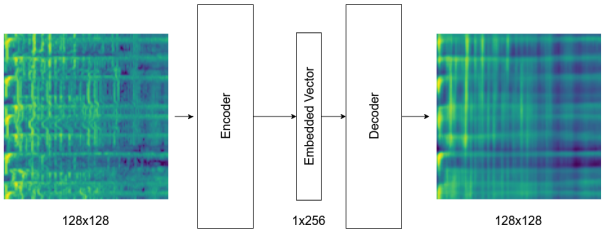

Figure 1. MEL-AE

El autoencoder esta compuesto de dos bloques principales, el encoder y el decoder.

4.1.1 Encoder. El encoder esta compuesto una cascada de *layers* de convolución 2D para obtener información mas abstracta, y *layers* de MaxPool para reducción de dimensionalidad. La última *layer* antes de llegar a la capa oculta (o el embedding) es una layer Fully Connected (con una capa de flatten previa), que nos permite llegar a la dimensión para nuestros *embeddings*.

4.1.2 Decoder. Al ser un autoencoder *vanilla*, el decoder no es nada mas que un espejo del encoder, cambiando las capas de MaxPooling usadas para reducción de dimensionalidad, por capas UpSampling que aumentan esta. Finalmente se agrega una *layer* de convolución 2D con un único filtro para llegar a una imagen profundidad unidimensional.

Puede encontrar en el anexo 6 el modelo de manera detallada

4.1.3 Entrenamiento de la red. La red fue entrenada usando todas las canciones presentes en el MelDataset (durante la etapa de validación de hiperparámetros se usaba la estrategia 9:1 test:validación). Para poder usar nuestra red, fue necesario reducir los espectrogramas originales, ya que estos aun si son siempre de 128 píxeles de altura, el ancho superaba esto considerablemente (la moda siendo de 1296 píxeles). Se procedió en cada época de entrenamiento, a obtener 5 fragmentos al azar de 128x128 para generar los *batches* de entrenamiento. Estos fragmentos se llamaran *patches* desde ahora en adelante. Se utilizó un optimizador Adam con *learning rate* de 0.0008 y *binary cross-entropy* como función de perdida. Se llegó a la convergencia después de 5 épocas de entrenamiento. Con la red ya entrenada, se pudo obtener los *embeddings* para espectrogramas de 128x128 píxeles.

4.2 Embeddings

4.2.1 Creación de los *embeddings*. Para generar los *embeddings*, se obtuvieron *patches* metódicamente, tal que definido el ancho de una ventana (de 128 píxeles), uno obtiene *patches* moviendo un *stride* determinado esta ventana desde el inicio de la imagen hasta el final de la imagen. El *stride* usado fue definido en $\alpha = 0.7$, esto significando que la ventana se mueve un 70% de su ancho en cada intervalo. Para los espectrogramas originales de 1296 píxeles de ancho, se obtuvieron 14 *patches*, lo que consecuentemente nos entrega 14 *embeddings* para cada canción (al correr nuestra red entrenada en cada *patch*).

4.2.2 Análisis de *embeddings*. Para determinar si los vectores latentes de los *patches* de las canciones generados por el modelo desarrollado representan correctamente el contenido musical, se utilizó el algoritmo tSNE (*t-Distributed Stochastic Neighbor Embedding* [15]), un algoritmo que reduce la dimensionalidad de los vectores intentando mantener las posiciones relativas, siendo bastante útil para la visualización de espacios semánticos complejos en un gráfico 2D:

Se puede observar en la figura 2 que existen *clusters* de *patches* en el espacio obtenido por el algoritmo tSNE, que corresponden a una misma canción. Esto nos entrega la idea de que el *embedding*, por lo mínimo mantiene algún sentido entre secciones de una canción.

No siendo esto suficiente para convencernos, tomamos una serie de *tracks* tales que sus centros de masa se localizaran cercanamente y usamos la API de Spotify junto a sus URI para obtener sus links y realizar un rápido testeo auditivo.

Se puede acceder a los links presentados en la tabla 1 para escuchar los *tracks* pintados en la figura 2, donde queda a la subjetividad humana el parecido entre estos *tracks*. Como

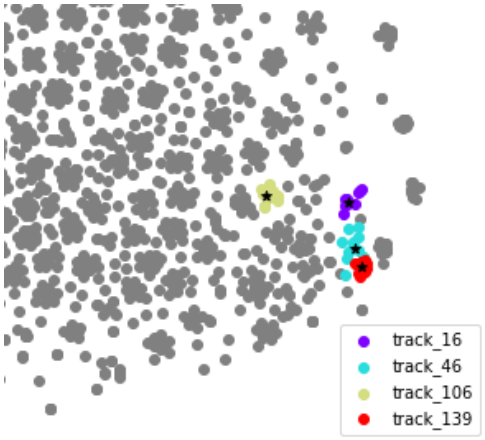

Figure 2. Visualización *embeddings* mediante tSNE. Cada punto representa un *patch*, y cada color representa una canción distinta. Las estrellas negras muestran el centro de masa de cada canción.

Table 1. Información *tracks* de la figura 2.

Track	Title	Artist	Sample
16	The Imitation Game	Alexandre Desplat	Link
46	Sorbetto	Billy Joel	Link
106	Divertissement	Saint-Preux	Link
139	Merry Christmas	Piano Christmas	Link

opinión de los autores, consideramos que las cuatro canciones pertenecen a música instrumental con ritmos parecidos, concluyendo que los *embeddings* obtenidos si representan de cierta manera el contenido musical de los *tracks*. Además, cabe notar que usar el promedio de los *patches* es una sencilla y buena forma de obtener los *embeddings* de cada canción.

Debe ser mencionado que este no es el caso para todo el *dataset*, ya que existen *tracks* que tienen sus *patches* muy separados o que *tracks* cercanos son de un estilo musical muy diferente. Para mayor información sobre este análisis revisar el Anexo.

4.3 Metodología de recomendación

Una vez obtenido los *embeddings* de cada *track*, se tuvo que investigar la mejor forma para generar recomendaciones a cada playlist. Dado que es un sistema basado en contenido, se procedió a utilizar una función de *score* como la que se presenta en [9]. Esta función consiste en que dado una playlist que contiene un set de *tracks* T_p y un ítem i de la biblioteca de música, el *score* que se le asigna al ítem i para ser recomendado a la playlist p es:

$$\text{score}(p,i)_X = \begin{cases} \max_{j \in T_p} \left\{ \text{sim}\left(V_i^X, V_j^X\right) \right\} & \text{(maximum)} \\ \frac{\sum_{j \in T_p} \text{sim}\left(V_i^X, V_j^X\right)}{|T_p|} & \text{(average)} \\ \frac{\sum_{r=1}^{\min\{K,|T_p|\}} \max_{j \in T_p}^{(r)} \left\{ \text{sim}\left(V_i^X, V_j^X\right) \right\}}{\min\{K,|T_p|\}} & \text{(average top}K\text{)} \end{cases} \tag{1}$$

, donde V_z^X es el vector latente (*embedding*) del ítem z, que para este caso es de dimensión $\mathcal{R}^{256}$. La función $sim(V_i^X, V_j^X)$ corresponde a la similaridad que hay entre el vector del ítem i y j. Al igual que en [9], se utilizó la similaridad coseno que corresponde a la siguiente ecuación:

$$\text{sim}\left(V_i, V_j\right) = \cos\left(V_i, V_j\right) = \frac{V_i \cdot V_j}{\|V_i\| \left\|V_j\right\|} \tag{2}$$

Por lo tanto, para la recomendación de cada playlist, se realiza la similaridad entre esta y la biblioteca musical, y se eligen los *top@K* según a uno de los métodos de la ecuación 1.

5 Experimentos

5.1 Datos

Se utilizó como base el MPD [6] y usando la API de Spotify, se pudo generar espectrogramas de un poco más de 60 mil canciones que corresponden a 5 mil playlists. Cabe notar que la API permitió obtener muestras de 30 segundos de aproximadamente 55% de las canciones de estas playlists. Luego, se utilizó el autoencoder para generar los *embeddings* de cada *track* y se filtraron las playlists que tenían un largo menor a 10 canciones. La tabla 2 resume el dataset utilizado.

Table 2. Resumen estadísticas del dataset.

Property	Value
Number of playlists	4100
Number of tracks	162,565
Number of unique tracks	59,410
Average playlist length (tracks)	37
Most popular track (ocurrence)	222

5.2 Baselines

Dado que se generó un propio dataset, resulta bastante complicado comparar resultados con otros modelos del estado del arte. En particular, dado la naturaleza del MPD y que sólo utilizamos 5000 playlists, resulta poco útil realizar recomendación colaborativa al tener una distribución con *long tail* como se aprecia en la figura 3. Sin embargo, un buen punto de partida para comparar resultados es contra *most popular*, ya que este generalmente entrega buenos resultados en tema de exactitud. Otra forma de ver si los *embeddings* generados por MelAE son más representativos que otros, se utilizaron las características que genera Spotify para cada canción para

crear un vector latente. Este *embedding* de dimensión 10 contiene características tales como *acousticness*, *danceability*, *loudness*, entre otras.

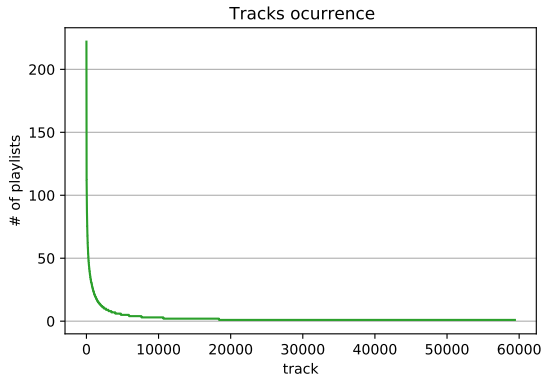

Figure 3. Distribución de los *tracks* del MelDatset.

5.3 Métricas

Dado que se va a realizar una recomendación de ranking, se utilizaron las métricas de exactitud MAP y nDCG. También, se van evaluó con otras métricas como *novelty* y *coverage*, en base a la definición entregada por [7], ya que al ser la música algo complejo de recomendar de forma off-line, la exactitud no lo es todo para decir que se generan buenas recomendaciones. Por un lado, *novelty* otorga un valor inversamente proporcional en base a la cantidad de ocurrencias que tienen los ítems recomendados dentro del dataset (en este caso las playlists). Por otro lado, *coverage* observa cuántos ítems son recomendados sobre el *dataset* completo. Además de estas métricas, se hizo una evaluación cualitativa de los resultados para ver si hacen sentido a un nivel intuitivo, ya que al fin y al cabo, este es el objetivo del modelo implementado.

5.4 Análisis método de *score*

Antes de comparar los modelos con las métricas descritas anteriormente, se hizo un análisis de cuál es el mejor método para recomendar según la ecuación de *score* (1). Además de los métodos mostrados 1, se añadió un cuarto llamado *cluster*. Este método consiste en realizar *clustering* para cada playlist y elegir el *cluster* más grande para realizar la similaridad entre ítems, con el fin de que los *outliers* no interfieran con las recomendaciones. Este experimento se hizo para *top*@5 recomendaciones con los *embeddings* de MelAe y el que contiene las características de Spotify.

Se puede ver claramente que las recomendaciones con mayor exactitud las genera el modelo MelAe con el método del máximo. En cambio, el método *top_k*3 tiene ligeramente mejor desempeño los vectores obtenidos de Spotify. Cabe notar, que se analizó también las otras métricas, pero no aportaban mayor información a la que se muestra en el gráfico anterior.

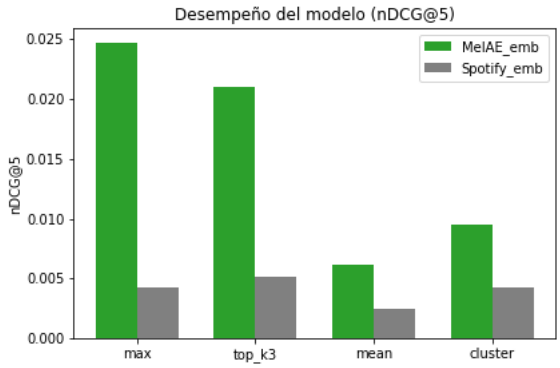

Figure 4. Comparación de métodos para recomendación *top*@5.

5.5 Resultados

Una vez elegido el mejor método de *score* para cada modelo, se pudo obtener el desempeño que tuvo cada modelo. Estos resultados se muestran en la tabla 3.

Table 3. Resultados para recomendaciones top@N.

Modelo	MAP	nDCG	Coverage	Novelty
Most_pop@5	**0.033**	**0.040**	0.000	4.54
MelAE@5	0.021	0.025	0.110	11.23
Spfy@5	0.004	0.005	**0.221**	**11.25**
Most_pop@10	**0.037**	**0.052**	0.000	4.65
MelAE@10	0.022	0.028	0.170	**11.27**
Spfy@10	0.005	0.007	**0.344**	11.26

Se puede ver que en métricas de exactitud, *most popular* supera a los otros dos modelos sin importar el *top@N*. Sin embargo, MelAE no tiene un desempeño tan bajo como el de Spfy. Luego, por definición, el método de *most popular* fracasa rotundamente en el *coverage*, mientras que Spfy tiene el doble del valor de MelAE. Cabe notar, que tener un alto *coverage* no necesariamente es bueno, ya que pueden ser indicios de que las recomendaciones son aleatorias. Por último, ambos modelos tiene un valor similar en términos de *novelty*, pero superan con creces a *most popular*, lo que es lógico nuevamente por la definición de este modelo.

5.6 Evaluación cualitativa

Como se dijo anteriormente, las métricas son buenos indicios para analizar si se creó un buen sistema recomendador. Sin embargo, hay muchos otros factores que puede influir para generar buenas recomendaciones que las evaluaciones cuantitativas no pueden medir. Por esto, y dado a que es

recomendación de música, lo más común es escuchar si los *tracks* recomendados tiene sentido y se adecuan a la playlist.

Para realizar esta evaluación, se presentan las playlist 715 y 4096 como ejemplos (se mostrarán sólo 10 canciones, debido a que algunas listas poseen más de 20 canciones). Las recomendaciones en negritas son las que se encontraban dentro de el set de evaluación o *ground truth*.

Table 4. Ejemplos recomendaciones para la playlist 715.

Track	Title	Artist	Sample
Playlist			
1	The Sound of Your Heart	Four Year Strong	Link
2	Eating My Words	Four Year Strong	Link
3	We All Float Down Here	Four Year Strong	Link
4	The Girls on Drugs	Wale	Link
5	Heres to Swimming With Bow Legged Women	Four Year Strong	Link
6	Wolverines	I Am The Avalanche	Link
7	No Role Modelz	J. Cole	Link
8	Love Yourz	J. Cole	Link
9	Im a Big, Bright, Shining Star	Four Year Strong	Link
10	Brooklyn Dodgers	I Am The Avalanche	Link
Recomendaciones			
1	**Gravity**	Four Year Strong	Link
2	**Who Cares?**	Four Year Strong	Link
3	**Wipe Yourself Off, Man. You Dead.**	Four Year Strong	Link
4	True Believers	The Bouncing Souls	Link
5	False Walls	Unleash The Archers	Link

Table 5. Ejemplos recomendaciones para la playlist 4096.

Track	Title	Artist	Sample
Playlist			
1	Sol Invictus	Audiomachine	Link
2	Wolverines	Ramin Djawadi	Link
3	Answer to No One	Colt Ford	Link
4	SkyWorld	Two Steps from Hell	Link
5	El Dorado	Two Steps from Hell	Link
6	Sunday	Les Friction	Link
7	Equinox	Audiomachine	Link
8	Ocean Princess	Thomas Bergersen	Link
9	For the Win	Two Steps from Hell	Link
10	Pompeii	E.S. Posthumus	Link
Recomendaciones			
1	Aura	Thomas Bergersen	Link
2	Dachuur	Two Steps from Hell	Link
3	**Love & Loss**	Two Steps from Hell	Link
4	**Victory**	Two Steps from Hell	Link
5	Neighbors	The Academy Is...	Link

Se puede apreciar que para la *playlist* 715, se esta recomendando un artista presente en la playlist original. Si se accede a los *links*, se podrá notar que las canciones de la playlist y las recomendadas son notablemente similares, siendo ambas del género hardcore-melódico.

Por otro lado, para la lista 4096, las primeras cuatro canciones recomendadas son del mismo género, al igual que la mayoría de las presentes en la lista original, siendo este género producción cinematográfica. La quinta recomendación, por otro lado, es rock, lo que se puede deber a la presencia de *Answer to No One - Colt Ford* canción en la lista original.

6 Conclusiones

El autoencoder que se utilizó para obtener los *embeddings* cumplió su objetivo de poder representar el contenido musical de los *tracks* en un vector de dimensionalidad reducida.

A pesar de que MelAE no tuvo la mejor exactitud en los experimentos, fue el modelo más regular considerando todas la métricas. El hecho de hacer sólo experimentos *off-line* impide de evaluar verdaderamente la calidad de las recomendaciones. No obstante, se pudo mostrar con ejemplos de recomendaciones que estas pueden ser adecuadas para la *playlist* elegida.

La elección de obtener el promedio de los *patches* para generar el *embedding* de cada *track* puede haber influido en la calidad de las recomendaciones, al mezclar o perder información relevante de la representación musical.

7 Trabajo futuro

El trabajo presentado se podría considerar en su forma mas básica, particularmente debido al uso de un modelo de autoencoder convolucional notablemente simple, así como algoritmos de *clustering* para la *representación* de playlist (o la falta de estos) y las posterior recomendación a base de similaridad coseno.

Para mejorar el rendimiento del sistema, se presentan variadas ideas como son:

7.1 End to End system

Las redes neuronales son considerablemente flexibles en la definición de su *loss function*. Una posible manera de mejorar la calidad de la recomendación es usar la información del comportamiento de los usuarios en la función de perdida del autoencoder.

De esta manera, el modelo intentaría generar un *embedding* que represente el contenido musical donde las distancias representen mejor los gustos de los usuarios. Lamentablemente, lograr esto consideraría expandir el modelo presentado considerablemente, ya que sería necesario que el entrenamiento de la red incluya todo el proceso hasta la recomendación, lo que trae consigo un problema de costo computacional y/o la necesidad de optimizar los pasos.

7.2 Contextualización de los tracks/ Multi-task learning

Nuestro modelo sólo trabaja con la idea de reconstruir la imagen original. En el caso de poseer información contextual de un *track* (ej. autor, álbum, lenguaje, país de origen, género,

etc), uno podría forzar a los *embeddings* a que *tracks* con información contextual parecida posean una distancia menor en el espacio embebido.

7.3 Metric learning

El proceso de recomendación usado usa distancia/similaridad coseno, usando el mínimo/máximo para generar el ranking. Esto considera al espacio semántico como un espacio euclidiano donde las distancias entre objetos es similar según la posición, pero considerando que este espacio fue generado para representar las características musicales de una pieza, *metric learning* se presenta como una posible mejora para evaluar la similaridad de canciones. Naturalmente, es necesario un tipo de información para llevar a cabo el entrenamiento de *metric learning*, como sería información contextual de las piezas.

7.4 Clustering

Al momento de hacer *playlist continuation*, las playlists fueron identificadas como una serie de puntos en el espacio embebido, siendo cada punto el centro de masa de cada canción. Considerando que cada pieza presenta una serie de puntos debido a sus *patches*, uno podría hacer uso de algoritmos de *clustering* como k-means o DBSCAN para identificar las ideas semánticas más abstractas presentes es una *playlist* y hacer una recomendación mas adhoc a estas.

References

[1] David Alexandre, Chih-Peng Chang, Wen-Hsiao Peng, and Hsueh-Ming Hang. 2019. An Autoencoder-based Learned Image Compressor: Description of Challenge Proposal by NCTU. arXiv:1902.07385 [cs.CV]

[2] Federico Baldassarre, Diego González Morín, and Lucas Rodés-Guirao. 2017. Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2. arXiv:1712.03400 [cs.CV]

[3] Pierre Baldi. 2012. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning (Proceedings of Machine Learning Research, Vol. 27)*, Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver (Eds.). JMLR Workshop and Conference Proceedings, Bellevue, Washington, USA, 37–49. http://proceedings.mlr.press/v27/baldi12a.html

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. 2020. Autoencoders. arXiv:2003.05991 [cs.LG]

[5] Òscar Celma. 2010. *The Long Tail in Recommender Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 87–107. https://doi.org/10.1007/978-3-642-13287-2_4

[6] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. Recsys Challenge 2018: Automatic Music Playlist Continuation. (2018), 527–528. https://doi.org/10.1145/3240323.3240342

[7] Marius Kaminskas and Derek Bridge. 2016. Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems. *ACM Transactions on Interactive Intelligent Systems* 7 (12 2016), 1–42. https://doi.org/10.1145/2926720

[8] Jongpil Lee, Kyungyun Lee, Jiyoung Park, Jangyeon Park, and Juhan Nam. 2018. Deep Content-User Embedding Model for Music Recommendation. arXiv:1807.06786 [cs.IR]

[9] Pablo Messina, Vicente Dominguez, Denis Parra, Christoph Trattner, and Alvaro Soto. 2019. Content-based artwork recommendation: integrating painting metadata with neural and manually-engineered visual features. *User Modeling and User-Adapted Interaction* 29, 2 (01 Apr 2019), 251–290. https://doi.org/10.1007/s11257-018-9206-9

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 [cs.CL]

[11] Sergio Oramas, Oriol Nieto, Mohamed Sordo, and Xavier Serra. 2017. A Deep Multimodal Approach for Cold-start Music Recommendation. *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems - DLRS 2017* (2017). https://doi.org/10.1145/3125486.3125492

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, David E. Rumelhart and James L. Mcclelland (Eds.). MIT Press, Cambridge, MA, 318–362.

[13] Markus Schedl. 2019. Deep Learning in Music Recommendation Systems. *Frontiers in Applied Mathematics and Statistics* 5 (2019), 44. https://doi.org/10.3389/fams.2019.00044

[14] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2643–2651. http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf

[15] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

[16] Guoqiang Zhong, Haizhen Wang, and Wencong Jiao. 2018. *MusicCNNs: A New Benchmark on Content-Based Music Recommendation: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I.* 394–405. https://doi.org/10.1007/978-3-030-04167-0_36

A Embeddings erróneos

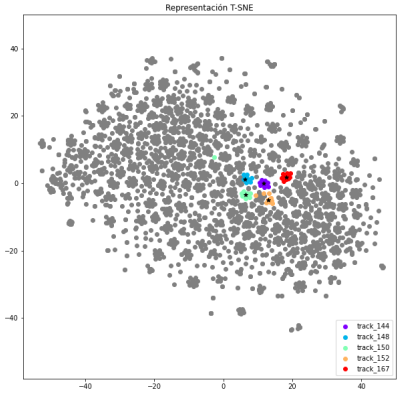

Figure 5. *Embeddings* con posición similar, pero distinto contenido musical.

Estos *tracks*, a pesar de tener posiciones similares en el espacio tSNE, son de géneros bien distintos, por lo que hace pensar que los vectores no son tan representativos. De igual

modo, cabe mencionar que tSNE es una reducción de dimensionalidad y se hizo sólo con 240 *tracks*, por lo que no es del todo representativo.

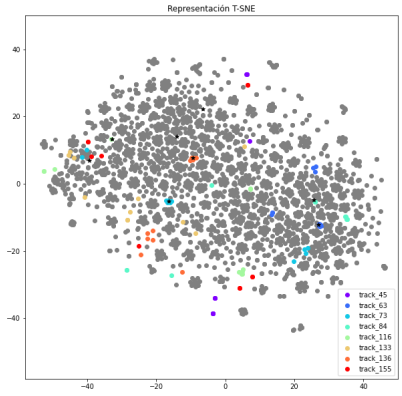

Figure 6. *Embeddings* con *pactches* muy separados en espacio tSNE.

Se puede ver que también existen *tracks* que tiene sus *patches* muy separados entre sí y no necesariamente es por tener contenidos musicales distintos, por lo que el promedio de estos no es una buena representación del *track*.

B Arquitectura del autoencoder

Table 6. Descripción detallada del modelo.

	Capas
Encoder	Input(128,128,1)
	Conv2D(3,3,16)
	MaxPool(2,2)
	Conv2D(3,3,8)
	MaxPool(2,2)
	Conv2D(3,3,16)
	MaxPool(2,2)
	Flatten
	Dense(256)
Embedding	1
Decoder	Dense(256)
	Reshape(16,16,16)
	Conv2D(3,3,16)
	UpSampling(2,2)
	Conv2D(3,3,8)
	UpSampling(2,2)
	Conv2D(3,3,16)
	UpSampling(2,2)
	Conv2D(3,3,1)