UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Motor de búsqueda semántico para canciones y poemas con integración de Spotify

Informe final - Recuperación de Información y Recomendaciones en la Web

Lucas Lazogue, Marcos Fierro, Ignacio Sastre, Gabriel Machin

Montevideo, 1 de julio de 2024

Índice

1. Introducción

A medida que la variedad de contenido digital se expande (situación que hace ya años está en constante crecimiento y cuyo ritmo de expansión es cada vez más vertiginoso), hay una considerable necesidad por desarrollar herramientas de búsqueda más sofisticadas que vayan más allá de la coincidencia de palabras clave y se adentren en la comprensión semántica de cada contenido. Situándonos específicamente en el entorno de la música, es muy probable que alguna vez todos hayamos "tarareado" una canción para intentar descifrar cuál era esa melodía que nos quedó grabada en la memoria aunque en escaso detalle. Si bien lo anterior es una manera bastante compleja de recuperar una canción, otro puede ser el caso en dónde recordemos aproximadamente de qué trataba la canción (en términos generales, no dictando textualmente una porción de la letra) y esto sería bueno nos permita encontrar la música que tenemos en mente. Otro caso diferente al anterior, es que inicialmente no conozcamos canciones que nos podrían resultar de gran interés si estamos buscando música que hable acerca de un tema puntual, como por ejemplo "la solidaridad" o "la familia" o "los amigos". Por tanto, un buscador semántico permitiría descubrir nuevas obras que nunca contemplamos y que se relacionen con el concepto, emoción o situación que describimos como punto de partida de la búsqueda.

Este proyecto pretende entonces desarrollar un motor de búsqueda inteligente que permita a los usuarios encontrar canciones y poemas basándose en las descripciones de sus temáticas, motivos o emociones. Tomando ventaja de las capacidades de procesamiento del lenguaje natural que hoy día se encuentran disponibles, y combinado con un almacenamiento y recuperación eficientes de representaciones vectoriales mediante PostgreSQL y su extensión pg-vector, la herramienta propuesta se integrará con Spotify de forma que los usuarios puedan recolectar rápidamente los resultados y convertirlos en listas de reproducción que pueden reproducir allí mismo o compartirlas con otras personas. Con lo anterior, se espera facilitar y mejorar la interacción del usuario con la música y la literatura.

2. Arquitectura del Sistema

Definimos la siguiente arquitectura para la aplicación, cuyo diagrama se muestra en la Figura 1. Esta arquitectura está compuesta por los siguientes componentes:

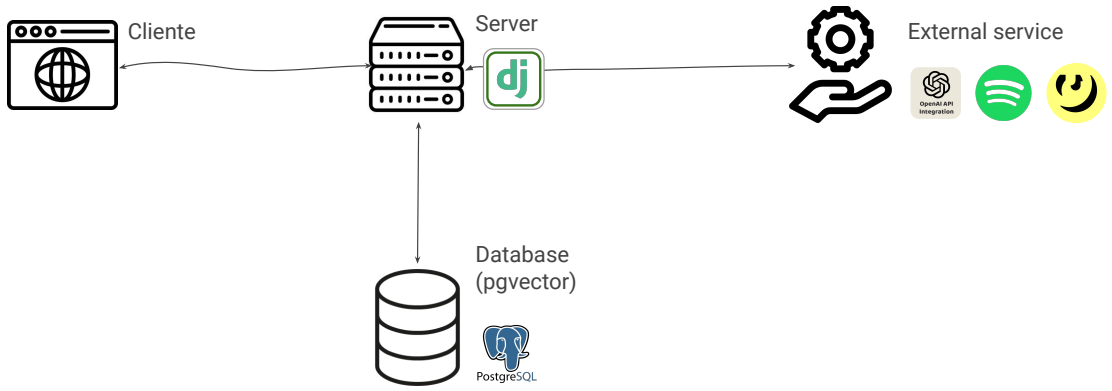

Figura 1: Arquitectura de la aplicación

1. **Servidor y Frontend:** En el núcleo del sistema se encuentra el servidor web desarrollado con el framework Django del lenguaje de programación Python. El mismo maneja la lógica de la aplicación y la generación de las vistas con los resultados. La interfaz web se desarrolla utilizando tecnologías estándar como HTML, JavaScript y CSS.

2. **Servicios Externos:** El servidor se complementa mediante la integración con varios servicios externos que son esenciales para las funcionalidades de la aplicación:

   - **OpenAI:** Utilizado para generar los *embeddings* de las búsquedas. OpenAI provee los servicios de procesamiento del lenguaje natural que permiten generar representaciones de la semántica del contenido de las canciones.
   - **Spotify:** Integrado para crear listas de reproducción basadas en los resultados de las búsquedas. La API de Spotify permite gestionar estas listas y sincronizarlas con la cuenta del usuario.

3. **Base de Datos:** La persistencia de datos se maneja a través de una base de datos PostgreSQL que incluye la extensión pg-vector. Este componente es fundamental para el almacenamiento y la consulta eficiente de los *embeddings* vectoriales generados por el modelo de OpenAI. Los datos almacenados incluyen información sobre las canciones, como título, artista, año de publicación, género y sus representaciones vectoriales.

3. Recolección y Preprocesamiento de Datos

Se utilizó el dataset *Genius Song Lyrics* [1], que contiene 5.134.668 canciones obtenidas mediante *scraping* del sitio web https://genius.com/, en donde se suben letras de las canciones, e incluso poemas, así como otros metadatos relevantes.

El dataset consiste de los siguientes campos: título de la obra, letra completa de la canción (o poema), género musical, artista, año de lanzamiento, cantidad de visitas, información sobre otros artistas que contribuyeron a la obra y el lenguaje de la letra analizado por dos modelos automáticos.

La Figura 2 muestra la distribución de las obras en los distintos géneros presentes en el dataset. El Cuadro 1 contiene información relevante sobre la composición del dataset.

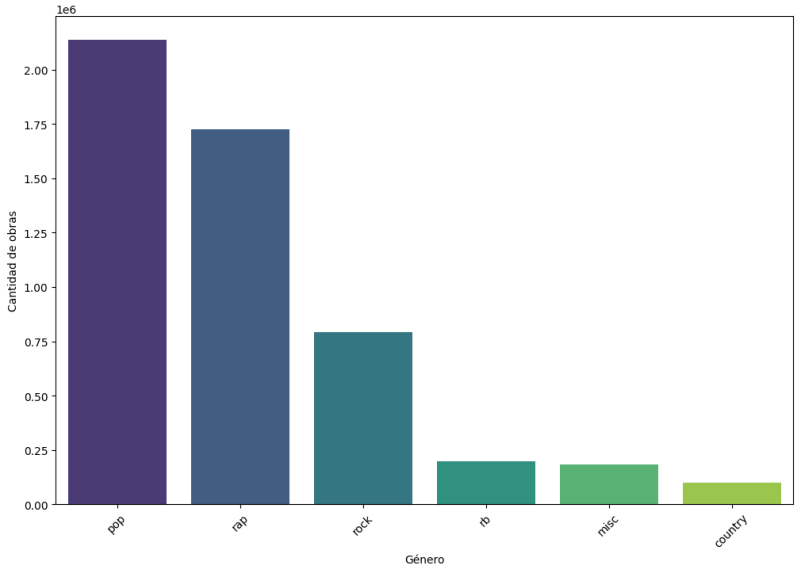

Figura 2: Distribución de las obras en los distintos géneros

Descripción	**Valor**
Artistas únicos	641349
Géneros únicos	6
Lenguajes únicos	84
Año de canción más antigua	1
Año de canción más reciente	2024
Cantidad total de obras	5134668
Cantidad total de visitas	15717481921
Letra más larga (en caracteres)	107132
Letra más corta (en caracteres)	100

Cuadro 1: Información sobre la composición del dataset Genius Song Lyrics

[1] https://www.kaggle.com/datasets/carlosgdcj/genius-song-lyrics-with-language-information

3.1. Selección de canciones

Se decidió tomar un subconjunto del dataset con el objetivo de ser utilizado como prueba de concepto en el prototipo a desarrollar. Se seleccionaron las 2000 obras en inglés con más visitas. Se decidió utilizar canciones solo en inglés para facilitar la búsqueda semántica, limitando el uso de la herramienta solo a este lenguaje en primera instancia.

Debido a que se utiliza el modelo de generación de embeddings de OpenAI para generar las representaciones vectoriales de las letras de las obras (ver Sección 4), es necesario que el largo máximo de las letras de las obras este restringido al largo máximo que la API de OpenAI permite enviar. Este largo está determinado por la cantidad de tokens, no la de caracteres. OpenAI disponibiliza una herramienta para contar la cantidad de tokens de un texto [2], que utilizamos para corroborar que las canciones más largas no excedieran el límite de 8192 tokens. Debido a esto, tuvimos que descartar una canción que excedía este límite, por lo que tomamos la siguiente canción con más visitas para completar las 2000 canciones totales.

La Figura 3 y el Cuadro 2 muestran la misma información que la Figura 2 y Cuadro 1 respectivamente, pero sobre el subconjunto de las 2000 canciones seleccionadas. Se puede observar que hay una gran representación de canciones del género rap, superando al pop, género más representado en todo el dataset. También es interesante notar que estas 2000 obras (0,04 % del total de obras) comprenden el 20,99 % de las visitas totales.

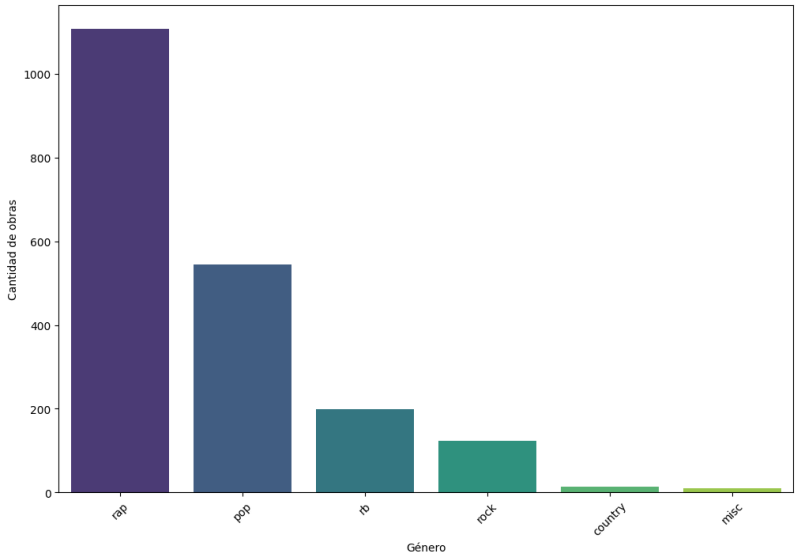

Figura 3: Distribución de las obras en los distintos géneros del subconjunto de 2000 canciones

[2] https://platform.openai.com/tokenizer

Descripción	Valor
Artistas únicos	705
Géneros únicos	6
Lenguajes únicos	1
Año de canción más antigua	1623
Año de canción más reciente	2022
Cantidad total de obras	2000
Cantidad total de visitas	3298437999
Letra más larga (en caracteres)	33183
Letra más corta (en caracteres)	166

Cuadro 2: Información sobre la composición del subconjunto de 2000 canciones

4. Sentence Embeddings

A continuación se presentará el método utilizado para realizar la búsqueda semántica sobre las letras de las obras.

4.1. Marco teórico

Los *embeddings* son representaciones vectoriales de pocas dimensiones (50-1000) y densos (valores mayoritariamente distintos de 0), que buscan capturar la semántica de una porción de texto, y se basan en la hipótesis distribucional, que establece que las palabras que son sinónimos tienden a ocurrir en los mismos contextos (*rodeadas* por las mismas palabras) (Jurafsky and Martin, 2009).

Mikolov et al. (2013) presentaron el algoritmo *word2vec*, utilizando redes neuronales, para obtener embeddings estáticos de las palabras de un lenguaje. Este método de representar las palabras mostró mejores resultados que los métodos clásicos de vectores dispersos basados en frecuencias.

Sin embargo, estas representaciones estáticas de las palabras tienen algunos problemas. Por ejemplo, no se tiene en cuenta el contexto en el que se encuentra la palabra, por lo que las palabras polisémicas tienen la misma representación a pesar de poseer múltiples significados. Además, no es sencillo representar una secuencia de texto con múltiples palabras (como una oración, o en nuestro caso la letra de una canción). Una alternativa es calcular el *centroide* (vector promedio) de los embeddings de las palabras, pero se pierde información de la posición de las palabras, así como la relación entre estas.

Con la aparición de la arquitectura Transformer (Vaswani et al., 2023) y, a partir del uso de esta arquitectura, la llegada de los modelos de lenguaje enmascarados como BERT (Devlin et al., 2019), se introdujo el concepto de embeddings contextualizados. Estas son representaciones vectoriales de las palabras, pero dependientes del contexto en el que se encuentran. Así, cada palabra no tiene una única representación, sino que se tiene una representación diferente por cada contexto en el que se encuentra.

Mediante el uso de estos modelos de lenguaje tipo BERT, también fue posible entrenarlos (mediante técnicas como *fine-tuning*) para generar embeddings de una secuencia de texto de largo arbitrario, llamados *sentence embeddings* (Reimers and Gurevych, 2019). Estos embeddings representan la semántica de toda la secuencia de texto, en contraposición con los *word embeddings* que representan la semántica de las palabras.

Podemos visualizar los sentence embeddings como puntos en un espacio n-dimensional (n es la cantidad de dimensiones de los embeddings), donde los puntos más cercanos entre sí corresponden a secuencias de texto semánticamente más similares y los más lejanos a secuencias semánticamente más diferentes.

4.2. Implementación

Para utilizar los sentence embeddings como método de búsqueda semántica sobre las letras de las obras se siguieron una serie de pasos. En primer lugar, cada una de las obras fue convertida a un embedding utilizando el modelo `text-embedding-3-small` [3] de OpenAI. Se generan representaciones vectoriales para las letras de las obras. Estos embeddings son almacenados en una base de datos vectorial. En nuestro caso, optamos por utilizar la extensión `pgvector` [4] de PostgreSQL, ya que se integra de forma natural a las consultas relacionales, las cuales utilizaremos para aplicar filtros sobre los metadatos (como género o artista).

Al momento en que un usuario realiza una consulta, se siguen los siguientes pasos:

1. Si el usuario seleccionó algunos de los filtros, estos son aplicados al dataset para reducirlo estrictamente.
2. Se genera el embedding de la consulta utilizando el mismo modelo de OpenAI.
3. Se calcula la similitud del coseno entre el embedding de la consulta y los embeddings de todas las obras.
4. Se rankean las obras en función de la similitud obtenida.
5. Se retornan las 10 obras más similares a la consulta.

Se calcula la similitud del coseno como métrica de distancia en lugar de la distancia euclídea debido a que es más eficiente y, dado que los vectores están normalizados, las métricas resultan equivalentes (el orden relativo que se obtiene es el mismo). Es fácil observar que la similitud del coseno para vectores normalizados se reduce a calcular el producto punto entre los dos vectores.

En términos de eficiencia, se puede observar que por cada consulta de usuario se necesita ejecutar una inferencia del modelo de lenguaje para generar el embedding de la consulta, y calcular m veces la similitud del coseno, donde m es la cantidad total de obras en la base de datos (en nuestro caso, $m = 2000$). A medida que m es más grande, este método se torna más costoso.

Para sortear esta dificultad, las bases de datos vectoriales ofrecen algoritmos de búsqueda aproximada (Approximate Nearest Neighbors - ANN) como son IVFFlat o HNSW (Hierarchical Navigable Small World) [5], que permiten obtener de forma muy eficiente los embeddings más similares perdiendo muy poco rendimiento.

[3] https://platform.openai.com/docs/guides/embeddings
[4] https://github.com/pgvector/pgvector
[5] https://tembo.io/blog/vector-indexes-in-pgvector

5. Aplicación y Experiencia de Usuario

5.1. Interfaz de Búsqueda

El frontend presenta una interfaz simple e intuitiva donde los usuarios pueden ingresar consultas de lenguaje natural y aplicar filtros como género y artista.

Figura 4: Frontend de la aplicación

A continuación, se describen los elementos numerados presentes en la imagen mostrada anteriormente para proporcionar una comprensión detallada de su estructura:

1. **Buscador a partir de lenguaje natural:** Ubicado en la parte superior, este campo permite a los usuarios realizar consultas mediante la introducción de texto en lenguaje natural, facilitando así una búsqueda más intuitiva y flexible.

2. **Filtro por género:** Este filtro proporciona una lista desplegable donde los usuarios pueden seleccionar un género musical, restringiendo los resultados al género de su interés.

3. **Filtro por artista:** Similar al filtro por género, esta opción permite a los usuarios especificar un artista para acotar la búsqueda a sus preferencias musicales particulares.

4. **Botón de búsqueda:** Este botón permite a los usuarios ejecutar la búsqueda con los filtros seleccionados, mostrando los resultados que cumplen con los criterios especificados.

5. **Grilla con resultados:** Los resultados de la búsqueda se presentan en una grilla organizada, exhibiendo detalles esenciales de cada canción o poema encontrado, como título, artista, género y año de publicación.

6. **Botón para exportar a Spotify:** Permite a los usuarios añadir las canciones devueltas por la consulta directamente a una lista de reproducción de Spotify, simplificando así el proceso de integración con su cuenta de música en streaming.

Esta estructura no solo optimiza la navegabilidad y usabilidad, sino que también asegura que los usuarios puedan encontrar y gestionar contenido de una manera rápida y efectiva.

5.2. Integración con Spotify

Tras recuperar los resultados de búsqueda, los usuarios tendrán la opción de añadir las canciones obtenidas por la búsqueda a una lista de reproducción de Spotify, al mismo tiempo que se brindará una lista preconfigurada con aquellos resultados que sean canciones. Esta integración se gestiona a través de la API de Spotify [6], que maneja la autenticación de usuarios y operaciones de lista de reproducción.

Al usuario elegir la opción de exportar las canciones a Spotify, se pide que el usuario autorice el acceso a ciertos recursos necesarios para la creación y manipulación de playlist, esto se logra utilizando OAuth 2.0 [7].

Luego de obtenido el permiso, se utiliza la metadata de las canciones, tales como el título y el autor de la misma para realizar la búsqueda en Spotify y así ser capaz de identificarlas en el servicio de streaming. Como último paso, se crea la playlist asociado al usuario del cual obtuvimos los permisos y se agregan las canciones con las identificaciones obtendias en el paso anterior.

6. Trabajo Futuro

A lo largo del desarrollo de la aplicación planteada, surgieron ideas de mejora así como también aspectos de la herramienta que pueden extenderse y presentar así un servicio más completo para el usuario final. Estos puntos fueron agrupados como trabajo pendiente a futuro que es posible realizar partiendo de la solución actual; a continuación se listan los mismos:

- Cachear los embeddings resultantes de los inputs de los usuarios.

  Como en todo sistema de consultas, algunas entradas son más frecuentes que otras ya sea por popularidad constante como por consecuencia de un suceso temporal o periódico (como quien quizás busca las canciones de los mundiales de fútbol cuando una nueva edición está a pocos días de comenzar). Conservar estos embeddings permiten ahorrar tiempo de cómputo y retornar resultados más rápido, a la vez que se evita el uso repetitivo de las APIs externas.

- Incluir en la interfaz del usuario la posibilidad de seleccionar una canción o poema listado como resultado de la consulta y acceder resumidamente a la letra del contenido para visualizar las secciones de esas letras que llevaron a seleccionar cada una de ellas.

- Mejoras en el filtro de Artista para que sea tolerante frente a errores de tipeo, de esta manera, si se escribe mal un nombre, o una letra falta (o se repite sin querer) el filtro lo detecte y

[6] https://developer.spotify.com/documentation/web-api
[7] https://datatracker.ietf.org/doc/html/rfc6749

corrija ese valor.

- Extender los filtros aplicados a metadatos:
  - Selección de múltiples géneros
  - Rango de años
  - Idioma

- Integración con YouTube Music y otros.

  Así como se implementó la integración con Spotify para poder crear listas de reproducción, también sería deseable ofrecer más alternativas de exportado de resultados, como lo puede ser YouTube Music o incluso crear un espacio de listas de reproducción interno a la aplicación y que los usuarios puedan compartir por medio de URLs públicas.

- Incorporar Poemas.

  Aunque desde el inicio la idea fué abarcar ambos tipos de contenido (canciones y poemas), tal como se mencionó previamente en la Sección 3.1, por motivos de alcance, se redujo el conjunto solo a canciones. Si bien el dataset utilizado posee algunos poemas, sería conveniente integrar otra fuente de datos especializada en este tipo de narrativa.

- Actualización continua de contenido.

  Actualmente la aplicación se nutre de un dataset estático que contiene canciones creadas hasta el momento en que se hizo el proceso de *scraping*. Sería conveniente poder alimentar la base de datos con canciones nuevas que se publican diariamente, para mantener la frescura de los datos.

7. Conclusión

Culminado el presente trabajo y alcanzadas las metas propuestas, se obtuvo una aplicación web, interactiva mediante una intuitiva interfaz, la cual acompañada de una oferta de contenidos reducida, soluciona los casos de uso que iniciaron la motivación para llevar a cabo esta prueba de concepto. El sistema logrado no solo permite al usuario adentrarse de una forma sencilla en la búsqueda de literatura y música, si no que también ofrece la posibilidad de obtener un resultado exportable como lo son las listas de reproducción, que permiten de manera instantánea disfrutar de los contenidos hallados.

Tras haber realizado varias pruebas, se concluye que la comparación vectorial que se lleva a cabo entre contenidos y consultas obtiene resultados aceptables. Si bien en algunos casos hubo algunas canciones de dudosa relación con lo requerido, esto es también esperable considerando que solamente se cuenta con 2000 canciones hasta aquí. A medida que el dataset crezca, otros contenidos pueden presentar mejores matcheos y así generar cada vez mejores listados. Las pruebas que se ejecutaron se agruparon principalmente en tres casos de uso, primero búsquedas de un concepto puntual, como "familia" o "amor", segundo las búsquedas de un estado de animo, emoción o situación en la que uno se puede encontrar (por ej, "recordar a alguien y desear volver a verlo") y finalmente, la búsqueda de un contenido describiendo lo que este narra en su letra pero indicándolo muy brevemente en lenguaje natural como por ejemplo "dos amigos se divierten bailando y disfrutan de actividades sencillas, en lugar de necesitar del dinero para pasar un buen rato" (canción de Sia - Cheap Thrills).

El trabajo conseguido deja abierta la puerta a muchas mejoras y posibilidades de extensión futuras, de la misma manera que dejó la satisfacción en el equipo de haber podido redondear la idea base y completarla con la creación de resultados utilizables.

Referencias

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, London [u.a.], 3. ed., [pearson international edition] edition.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.