

**EXPLORACIÓN DE CONTENIDOS MULTIMEDIA
BASADA EN LA SIMILITUD**

**SIMILARITY-BASED BROWSING OF MULTIMEDIA
CONTENTS**



**TRABAJO FIN DE GRADO
CURSO 2021-2022**

AUTORES

BORJA ADAY GUADALUPE LUIS
ÓSCAR MORUJO FERNÁNDEZ
RAFAEL NOBLEJAS PÉREZ
JHEISON ORELLANA DÍAZ
DIEGO ENRIQUE DE MIGUEL LÓPEZ

DIRECTOR

JOSÉ LUIS SIERRA RODRÍGUEZ

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Autorización de difusión

Los abajo firmantes, matriculados en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: “Exploración de contenidos multimedia basada en la similitud”, realizado durante el curso académico 2021-2022 bajo la dirección de José Luis Sierra Rodríguez en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Borja Aday Guadalupe Luis,

Óscar Morujo Fernández,

Rafael Noblejas Pérez,

Jheison Orellana Díaz,

Diego Enrique de Miguel López

14 de septiembre de 2022

Índice

Autorización de difusión	II
Índice	III
Índice de figuras	VI
Resumen	IX
Palabras clave	IX
Abstract	X
Keywords	X
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Plan de trabajo	2
1.4. Estructura del documento	3
1. Introduction	4
1.1. Motivation	4
1.2. Objectives	4
1.3. Workplan	5
1.4. Document structure	6
2. Trasfondo	7
2.1. Introducción	7
2.2. <i>Transformers</i>	12
2.2.1. Arquitectura	12
2.2.1.1. Codificación posicional	15
2.2.1.2. Mecanismo de atención	16
2.2.1.3. <i>Multi-Head Attention</i>	19
2.2.1.4. Normalización	21
2.2.1.5. Red neuronal densa	22
2.2.2. Resultados	24
2.2.3. <i>Transformers</i> : BERT	25
2.2.3.1. Preentrenamiento	28
2.2.3.2. Ajuste fino	29
2.2.3.3. Similitud semántica con BERT: <i>Cross-Encoders</i> y <i>Bi-Encoders</i>	30
2.2.4. <i>Bi-Encoders</i> mediante <i>Sentence-Transformers</i> : SBERT	32
3. Arquitectura del proyecto	35
3.1. SBMC: <i>Similarity-based browser of multimedia content</i>	35
3.1.1. Introducción	35
3.1.2. Arquitectura	37

3.1.3.	Inicialización del entorno y carga de datos	39
3.1.4.	Cálculo de los <i>sentence-embeddings</i> : <i>Bi-Encoder</i>	41
3.1.5.	Cálculo de las similitudes: <i>Faiss</i>	44
3.1.6.	Reclasificación de soluciones: <i>Cross-Encoder</i>	46
3.1.7.	Exploración de la matriz de similitud	48
3.1.8.	Exportación de datos	48
3.1.9.	Exploración en tiempo real	49
3.1.10.	Exploración con otro conjunto de datos	50
3.2.	Explorador de películas basado en la similitud	51
3.2.1.	Introducción	51
3.2.2.	Arquitectura	51
3.2.3.	Extracción de datos	52
3.2.4.	Preprocesamiento de datos	53
3.2.5.	Obtención de similitudes usando SBMC	55
3.2.6.	Entrenamiento de modelos	56
3.2.6.1.	Primera versión	56
3.2.6.2.	Segunda versión	57
3.2.6.3.	Tercera versión	58
3.2.6.4.	Conclusiones sobre el entrenamiento	60
3.2.7.	Base de datos	60
3.2.7.1.	Estructura de la base de datos	60
3.2.8.	Servidor	62
3.2.8.1.	Puntos de acceso	62
3.2.8.2.	Conector con la base de datos	64
3.2.8.3.	Ejecución del algoritmo en tiempo real	65
3.2.9.	Interfaz de usuario	65
3.2.9.1.	Implementación	66
3.2.9.2.	Estructura de la interfaz	70
3.2.9.3.	Funcionalidades	73
3.2.10.	Ejemplos de uso	74
4.	Aplicaciones prácticas	81
4.1.	Exploración de películas	81
4.2.	Exploración de artículos de investigación	88
4.3.	Exploración de software en base a su descripción	90
4.4.	Detección de tweets con sentimiento depresivo	92
5.	Conclusiones y trabajo futuro	95
5.1.	Conclusiones	95
5.2.	Trabajo futuro	96
5.2.1.	Trabajo futuro en el SBMC	96
5.2.2.	Trabajo futuro en el explorador de películas	97
5.	Conclusions and future work	98
5.1.	Conclusions	98
5.2.	Future work	99
•	Future work in SBMC	99
•	Future work in movie explorer	100
Contribuciones individuales al proyecto	101	

Rafael Noblejas Pérez	102
Jheison Orellana Díaz	104
Borja Aday Guadalupe Luis	105
Óscar Morujo Fernández	107
Diego Enrique de Miguel López	109
Apéndice	111
Generación de la base de datos	111
Inicialización del servidor	112
Inicialización de la aplicación web	113
Bibliografía	114

Índice de figuras

Figura 1. Imagen generada por DALL·E 2 con el texto “Gran Vía, Madrid like a futuristic city of 3048”.....	8
Figura 2. Métodos para medir la similitud semántica.....	9
Figura 3. Ejemplo de operación sobre los word-embeddings.....	10
Figura 4. Arquitectura de ELMo.....	11
Figura 5. Arquitectura del Transformer a alto nivel.....	13
Figura 6. Arquitectura original del Transformer.....	13
Figura 7. Ejemplo de traducción durante la predicción de la segunda palabra de la frase en inglés.....	14
Figura 8. Ejemplo de codificación posicional.....	16
Figura 9. Representación gráfica del resultado de aplicar self-attention sobre dos ejemplos.....	16
Figura 10. Matriz de atención.....	17
Figura 11. Función de atención “Scaled Dot-Product Attention”.....	18
Figura 12. Representación de la función “Scaled Dot-Product Attention”.....	19
Figura 13. Mecanismo de Multi-Head Attention.....	20
Figura 14. Batch Normalization vs. Layer Normalization.....	21
Figura 15. Representación del tipo de patrones detectados en diferentes capas de un Transfomer.....	23
Figura 16. Ejemplos de patrones que capturan diferentes keys en el estudio.....	23
Figura 17. Parámetros de Transformers base y Transformers big.....	24
Figura 18. Puntuación de diferentes modelos con el algoritmo BLEU.....	24
Figura 19. Diferentes tipos de tareas que BERT puede realizar.....	26
Figura 20. Arquitectura interna de BERT.....	27
Figura 21. Esquema de preentrenamiento y ajuste fino de BERT.....	29
Figura 22. Cross-Encoder durante la inferencia de similitud.....	30
Figura 23. Bi-Encoder vs. Cross-Encoder.....	31
Figura 24. SBERT durante el entrenamiento y la inferencia.....	33
Figura 25. Tabla de rendimientos de diferentes modelos en generación de sentence-embeddings.....	34
Figura 26. Captura de los modelos más descargados en Hugging Face.....	36
Figura 27. Información del sistema (GPU + RAM) asignado a cuenta gratuita.....	36
Figura 28. Información del sistema (GPU + RAM) asignado a cuenta Pro.....	37
Figura 29. Componentes del sistema.....	38
Figura 30. Arquitectura del cálculo de similitudes de la herramienta SBMC.....	39
Figura 31. Instalación de dependencias e inicialización de la GPU.....	40
Figura 32. Inicialización del directorio de trabajo y carga de datos.....	40
Figura 33. Inicialización del modelo.....	41
Figura 34. Cálculo o carga de los sentence-embeddings.....	43
Figura 35. Cálculo de similitudes: Faiss, carga de similitudes, y exploración en el espacio.....	45
Figura 36. Esquema de reclasificación de soluciones.....	46

Figura 37. Reclasificación o carga de similitudes: Cross-Encoder	47
Figura 38. Exploración de la matriz de similitud.....	48
Figura 39. Exportación de datos a una base de datos.	48
Figura 40. Exploración en tiempo real.....	49
Figura 41. Exploración con otro conjunto de datos.....	50
Figura 42. Celdas encargadas de la obtención de datos.....	52
Figura 43. Borrado de argumentos inválidos.....	53
Figura 44. Sinopsis no validas encontradas durante el preprocesamiento del conjunto.	54
Figura 45. Comparación de modelos con distinto hardware.	55
Figura 46. Domain Transfer.....	56
Figura 47. Datasets disponibles en Hugging Face.....	57
Figura 48. Pasos del método GenQ.	57
Figura 49. Ilustración del método MultipleNegativeRankingLoss.....	58
Figura 50. Pasos del método GPL.	59
Figura 51. Ejemplo GPL vs. QGen.....	59
Figura 52. Diagrama de la base de datos.	61
Figura 53. Versión 0, página principal.	66
Figura 54. Versión 0, vista detallada de una película.	67
Figura 55. Versión 1, página principal.	67
Figura 56. Versión 1, vista detallada de una película.	67
Figura 57. Versión 2, página principal.	68
Figura 58. Versión 2, búsqueda de una película.	68
Figura 59. Versión 2, vista detallada de una película.	68
Figura 60. Versión 3, página principal.	69
Figura 61. Versión 3, filtro de géneros.	69
Figura 62. Versión 3, filtro de género seleccionado.	69
Figura 63. Vista de películas sugeridas con paginación.	74
Figura 64. Botones de paginación.....	74
Figura 65. Vista de información de una película y sus similares.	75
Figura 66. Búsqueda de películas por título mediante consulta.	76
Figura 67. Búsqueda de películas por título mediante el algoritmo.	76
Figura 68. Búsqueda de películas por actores mediante consulta.....	77
Figura 69. Búsqueda de películas por actores mediante el algoritmo.	77
Figura 70. Búsqueda de películas por texto.	78
Figura 71. Búsqueda de películas con filtros de género.	79
Figura 72. Vuelta a la página principal.....	80
Figura 73. Películas similares en sinopsis a The Lord of the Rings: The Return of the King.....	81
Figura 74. Resultados obtenidos con el texto “A woman seeks revenge”.....	82
Figura 75. Resultados obtenidos con el texto “People locked in a cube in which there are traps”.	83
Figura 76. Resultados obtenidos con el texto “Spiderman”.	84
Figura 77. Resultados obtenidos con el texto “Lord of rings”.....	85
Figura 78. Resultados obtenidos con el texto “Starwars”.....	85
Figura 79. Resultados obtenidos con el texto “Brad Pitt, Angelina Jolie”.	86

Figura 80. Argumentos similares obtenidos con un texto proporcionado por parámetro.....	86
Figura 81. Argumentos similares obtenidos con un texto proporcionado por parámetro.....	87
Figura 82. Abstracts de artículos similares obtenidos con un texto proporcionado por parámetro.....	88
Figura 83. Títulos de artículos similares obtenidos con un texto proporcionado por parámetro.....	89
Figura 84. Títulos de artículos similares a otro artículo del conjunto.....	89
Figura 85. Descripciones de aplicaciones similares obtenidas con un texto proporcionado por parámetro.....	90
Figura 86. Descripciones de aplicaciones similares obtenidas con un texto proporcionado por parámetro.....	91
Figura 87. Descripciones de aplicaciones similares a otra dada.....	91
Figura 88. Arquitectura del clasificador de tweets depresivos.....	92
Figura 89. Matriz de confusión resultante.....	93
Figura 90. Métricas obtenidas a partir de la matriz de confusión.....	94

Resumen

Hoy en día existen muchas compañías y páginas web que hacen uso de algoritmos de Inteligencia Artificial para mejorar la experiencia que recibe el usuario final. En muchos casos se desconoce la naturaleza exacta de dichos algoritmos, pero lo que sí es innegable es lo potentes que son y el gran impacto que llegan a tener en los usuarios. Por ello, se ha desarrollado una herramienta que permite realizar una exploración en base a la similitud semántica de textos en un conjunto de datos. Es importante resaltar que la herramienta se ha desarrollado para que pueda funcionar con diferentes tipos de textos y de idiomas para hacer la herramienta lo más independiente posible de los datos. Para poner a prueba esta herramienta, se ha desarrollado una aplicación web que permite realizar una exploración sobre una colección de películas recogidas de la web de TMDB¹ comparando sus argumentos semánticamente.

Palabras clave

Explorador de Contenidos, Faiss, Procesamiento de Lenguaje Natural, Similitud Semántica, SBERT, Transformers.

¹ <https://www.themoviedb.org/?language=es>

Abstract

Nowadays there are many companies and websites that make use of Artificial Intelligence algorithms to improve the experience received by the end user. In many cases the exact nature of these algorithms is unknown, but what is undeniable is how powerful they are and the great impact they have on users. For this reason, a tool has been developed to perform an exploration based on the semantic similarity of texts in a dataset. It is important to highlight that the tool has been developed to work with different types of texts and languages in order to make the tool as data independent as possible. To test this tool, a web application has been developed that allows to perform a scan over a collection of movies collected from the TMDB website by comparing their arguments semantically.

Keywords

Content Explorer, Faiss, Natural Language Processing, Semantic Similarity, SBERT, Transformers.

1. Introducción

1.1. Motivación

Desde el 2012, el Aprendizaje Automático está en auge gracias a la inmensa cantidad de datos en internet y los avances en hardware, en concreto las GPU (del inglés *Graphics Processing Unit*) que favorecen la computación en paralelo. Así, empiezan a aparecer modelos de procesamiento de lenguaje natural, pero estos necesitan de datos etiquetados para entrenarse.

En 2017 con la aparición de los *Transformers* [1], aparecen nuevos modelos capaces de realizar un aprendizaje auto supervisado. Así, sin necesidad de tener los datos etiquetados, la cantidad de datos disponible en internet es inmensa, por lo que se fue aumentando los datos con los que se entrenaban estos modelos, dando lugar a la era de los enormes modelos del lenguaje.

Utilizando estos modelos del lenguaje y siendo conscientes de la gran magnitud de datos a los que, en su día a día, se puede enfrentar una persona de cara a la exploración de contenidos, con este trabajo queremos ofrecer a todo tipo de usuarios un entorno de fácil usabilidad que les permita explorar diversos conjuntos de datos midiendo la similitud semántica entre los textos que describen dichos datos.

1.2. Objetivos

Este trabajo tiene como objetivo principal la creación de un entorno funcional que dada una colección de datos multimedia sea capaz de relacionar los componentes de dicha colección entre sí mediante el uso de técnicas PLN [2] (Procesamiento de Lenguaje Natural) basadas en *Transformers* aplicadas sobre las descripciones textuales de dichos datos.

De esta forma, con un conjunto de datos se puede realizar una exploración basada en la similitud semántica de un atributo textual que describe los datos, como por ejemplo:

- El argumento de una película en un conjunto de datos de películas.
- El resumen de un libro en un conjunto de datos de libros.
- La descripción de una App en un conjunto de datos de aplicaciones.
- Reclamaciones de clientes en un conjunto de datos de atención al cliente de una empresa.
- Titular de una noticia en un conjunto de datos de noticias.

Más concretamente, el objetivo es crear una herramienta que permita realizar esta exploración de manera rápida, sencilla y eficaz, combinando las últimas técnicas del PLN. Se podrá realizar una exploración en tiempo real con una búsqueda del usuario y simplemente ir explorando el conjunto de datos con sus consultas; o procesar el conjunto entero y obtener la matriz de similitud en diferentes formatos y preparada para su uso en otro lugar, como por ejemplo en una base de datos para consultas web.

Esta herramienta combinará el uso de modelos conocidos como *Bi-Encoders* [3] para realizar la codificación del texto y modelos *Cross-Encoders* [3] para obtener un nuevo orden para los elementos recuperados con el primero. Esto es porque el primero es más rápido, aunque menos exacto, mientras el segundo es muy lento, pero más exacto como se verá más adelante. Además, la herramienta hará uso de nuevas técnicas especializadas en reducción de dimensionalidad y en cálculos de similitudes utilizando técnicas como *Faiss*¹ [4] que, como se verá, implementa varias técnicas de indexación y compresión de vectores que hacen uso del paralelismo que permite la GPU para hacer nuestra herramienta mucho más rápida.

Así, para poner en prueba nuestra herramienta, usaremos como punto de partida una colección de películas, largometrajes y cortos obtenidos gracias a la API pública que nos ofrece la página web de TMDB. A partir de esta colección se crea un entorno en forma de aplicación web, que obtendrá los datos resultantes de procesar la colección mediante llamadas a una base de datos en la que estarán almacenados.

Esta aplicación web permitirá al usuario final navegar entre las distintas entradas de la colección usando por ejemplo el título, el argumento de la película o actores entre otros.

1.3. Plan de trabajo

El desarrollo del proyecto se puede dividir en las siguientes fases:

- **Estudio del estado de la cuestión:** la primera fase del proyecto dedicada a la investigación de anteriores estudios y trabajos relacionados con el cálculo de similitudes y el uso de *Transformers*.
- **Desarrollo del motor de cálculo de similitud:** segunda fase dedicada al desarrollo del algoritmo de cálculo de similitudes, por los dos miembros encargados del *backend*.
- **Desarrollo de la interfaz web:** tercera fase dedicada al desarrollo de la aplicación web, por los tres miembros encargados del *frontend*.

¹ <https://faiss.ai>

- **Obtención de resultados y conclusiones:** cuarta fase del proyecto dedicada al testeo del algoritmo y de la interfaz web, y la consecuente obtención de resultados y conclusiones.

1.4. Estructura del documento

Esta memoria dedicada al proyecto se puede dividir en los siguientes apartados:

1. **Introducción:** resumen del proyecto, motivaciones, objetivos, y plan de trabajo.
2. **Trasfondo:** estudio del estado del arte de la cuestión y los distintos enfoques aplicados a estudios anteriores.
3. **Arquitectura del proyecto:** exposición de la arquitectura general del trabajo realizado: el algoritmo de exploración, la base de datos, el servidor, y la interfaz web.
4. **Aplicaciones prácticas:** exposición de los distintos resultados obtenidos al aplicar la herramienta desarrollada sobre diversos conjuntos de datos.
5. **Conclusiones y trabajo futuro:** conclusiones extraídas del proyecto y de posibles ideas futuras a implementar.

1. Introduction

1.1. Motivation

Since 2012, Machine Learning is booming thanks to the immense amount of data on the Internet and advances in hardware, in particular GPUs. Thus, natural language processing models begin to appear, but they need labeled data to be trained.

In 2017 with the appearance of Transformers [1], new models capable of performing self-supervised learning appear. Thus, without the need to have the data labeled, the amount of data available on the internet is immense, so the data with which these models were trained was increasing, giving rise to the era of huge language models.

Using these language models and being aware of the large amount of data that a person may have to deal with on a daily basis when exploring content, with this work we want to offer all types of users an easy-to-use environment that allows them to explore different data sets by measuring the semantic similarity between the texts that describe these data.

1.2. Objectives

The main objective of this work is the creation of a functional environment that, given a collection of multimedia data, is able to relate the components of such collection to each other by using NLP (Natural Language Processing) techniques based on Transformers applied on the textual descriptions of such data.

This way, with a dataset, an exploration can be performed based on the semantic similarity of a textual attribute describing the data such as, for example:

- The plot of a movie in a movie dataset.
- The summary of a book in a book dataset.
- The description of an App in an App dataset.
- Customer complaints in a company's customer service dataset.
- Headline of a news item in a news dataset.

More specifically, the goal is to create a tool that allows to perform this exploration in a fast, simple, and efficient way, combining the latest NLP techniques. It will be possible to perform a real-time exploration with a user search and simply browse the dataset with its queries; or process the entire dataset and obtain the similarity

matrix in different formats and ready for use elsewhere, such as in a database for web queries.

This tool will combine the use of models known as Bi-Encoders [3] to perform the text encodings; and Cross-Encoders [3] models to obtain a new order for the elements retrieved with the former. This is because the former is faster, although less accurate, while the latter is very slow, but more accurate as will be seen later. In addition, the tool will make use of new techniques specialized in dimensionality reduction and similarity calculations using techniques such as *Faiss* [4] which as will be seen implements several vector indexing and compression techniques that make use of the parallelism allowed by the GPU to make our tool much faster.

Thus, to test our tool, we will use as a starting point a collection of movies, feature films, and short films obtained thanks to the public API provided by the TMDB website. From this collection we create an environment in the form of a web application, which will obtain the data resulting from processing the collection through calls to a database in which they will be stored.

This web application will allow the end user to navigate between the different entries of the collection using for example the title, the plot of the movie or actors, among others.

1.3. Workplan

The development of the project can be divided into the following phases:

- **Study of the state of the art:** the first phase of the project dedicated to the investigation of previous studies and works related to similarity calculation and the use of Transformers.
- **Development of the similarity calculation engine:** second phase dedicated to the development of the similarity calculation algorithm, by the two members in charge of the backend.
- **Development of the web interface:** third phase dedicated to the development of the web application, by the three members in charge of the backend.
- **Obtaining results and conclusions:** fourth phase of the project dedicated to the testing of the algorithm and the web interface, and the consequent achievement of results and conclusions.

1.4. Document structure

This report dedicated to the project can be divided into the following sections:

1. **Introduction:** summary of the project, motivations, objectives, and workplan.
2. **Background:** study of the state of the art of the issue and the different approaches applied to previous studies.
3. **Project architecture:** presentation of the general architecture of the work carried out: the exploration algorithm, the database, the server, and the web interface.
4. **Practical applications:** presentation of the different results obtained by applying the tool developed on various data sets.
5. **Conclusions and future work:** conclusions drawn from the project and possible future ideas to be implemented.

2. Trasfondo

2.1. Introducción

En un mundo moderno como el actual en el que tenemos casi todo al alcance de un solo clic, preguntarle a un asistente de voz qué día es o qué tiempo hará la próxima semana son tareas que nos parecen muy sencillas. Sin embargo, esto no ha sido siempre así y es que el hecho de que un computador sea capaz de comprender el lenguaje con el que hablamos los humanos es bastante reciente. Todas estas tareas se pueden realizar hoy día gracias al nacimiento del Procesamiento del Lenguaje Natural, que sigue evolucionando.

“El PLN es el campo de la informática y la IA dedicado al estudio de métodos y técnicas para el tratamiento automático de datos lingüísticos, su objetivo principal es permitir que los ordenadores sean capaces de interpretar y comprender el lenguaje humano de forma natural, es decir, de la misma forma en que lo hacen los seres humanos. Se basa en el estudio de la lingüística computacional, la cual a su vez se divide en varias subdisciplinas, como la morfología, la sintaxis y la semántica. El PLN es un campo en constante evolución, y cada vez se están descubriendo nuevos métodos y técnicas para mejorar la forma en que los ordenadores pueden interpretar y comprender el lenguaje humano.”

El párrafo anterior ha sido generado automáticamente utilizando como entrada el texto “*El PLN es el*” con GPT-3¹ [5], uno de los mayores modelos del lenguaje que existen en la actualidad, lo que demuestra el potencial que tienen estas técnicas en auge.

En un principio los primeros sistemas se basaban en un gran conjunto de reglas. Más tarde, aparecieron nuevos enfoques con la ayuda del Aprendizaje Profundo que fueron inicialmente desechados por la falta de potencia computacional del momento. Hoy en día, gracias a los avances en hardware y en campos como la inteligencia artificial, estos mecanismos vuelven a estar en auge debido a su alta eficiencia y los buenos resultados que proporcionan.

El PLN no está centrado en una sola tarea, pues es aplicable a infinidad de ámbitos como pueden ser: asistentes de voz, generación de texto, *bots* conversacionales, análisis de sentimientos, búsqueda de similitudes entre textos...

Hasta ahora, al buscar una imagen en internet, Google muestra lo más parecido que encuentra en su base de datos. Sin embargo, con la aparición reciente de DALL·E 2² [6], Google podría ser capaz de entender lo que el usuario está buscando y generar la imagen que busca, sin necesidad de que ésta existiera previamente.

¹ <https://openai.com/api>

² <https://openai.com/dall-e-2>

Podemos ver un ejemplo en la Figura 1. Del mismo modo, recientemente se ha presentado COPILOT¹ [7], una herramienta que permite generar código explicándole con lenguaje natural lo que se quiere programar, siendo así una herramienta muy potente en la que apoyarse a la hora de desarrollar código.



Figura 1. Imagen generada por DALL-E 2 con el texto “Gran Vía, Madrid like a futuristic city of 3048”.

Fuente: <https://openai.com/dall-e-2/>

De todas las tareas que engloba el PLN, en este proyecto nos centraremos en la exploración de contenidos basada en la similitud semántica haciendo uso del STS (del inglés *Semantic Textual Similarity*), para así valorar la similitud del significado entre textos. En la Figura 2 se pueden ver diferentes métodos para medir la similitud semántica (véase [8] para más detalle).

El principal objetivo de estudio de este documento son enfoques basados en redes neuronales, más concretamente en el estado del arte de estos para el procesamiento del lenguaje natural: los *Transformers*.

¹ <https://openai.com/blog/openai-codex>

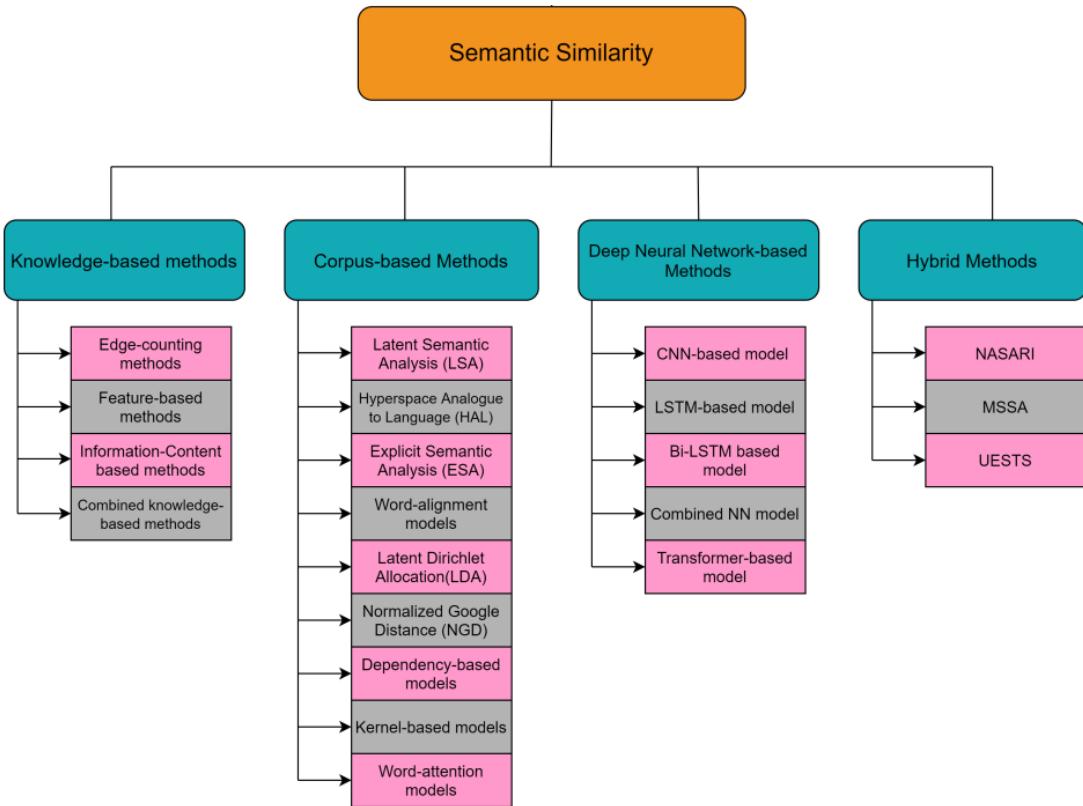


Figura 2. Métodos para medir la similitud semántica.

Fuente: <https://dl.acm.org/doi/10.1145/3440755>

Los ordenadores trabajan mejor con números y por ello los modelos de inteligencia artificial que trabajan con texto transforman las palabras de este a vectores numéricos para poder medir la distancia entre estos y así estudiar su similitud. Esta transformación a un espacio vectorial se puede hacer de diferentes maneras, como por ejemplo mediante *bag of words*¹ o bolsa de palabras en el que se tiene un vector de tantas posiciones como palabras hay en el vocabulario y cuyos valores pueden ser:

- **Número binario:** indica la presencia (1) o ausencia (0) del término.
- **Frecuencia del término [9]:** indica el número de apariciones del término dividido entre el número de términos totales del texto.
- **Frecuencia del término - Frecuencia de documentos inversa [9]:** multiplica la frecuencia del término por el valor de la frecuencia de documentos inversa, que mide la relevancia del término en el texto haciendo que uno que aparece en muchos textos tenga menos relevancia en la representación.

¹ https://en.wikipedia.org/wiki/Bag-of-words_model

Sin embargo, estos métodos basados en el corpus (*Corpus-based methods*) tienen un trasfondo estadístico y se adaptan bien a diferentes lenguajes, pero no son capaces de representar ni la semántica ni el orden de las palabras en un texto, por lo que estas técnicas no son de interés para el desarrollo de este proyecto.

Como se ha visto en la Figura 2, existen otros métodos: los métodos basados en conocimiento o reglas (*Knowledge-based methods*), que tienen en consideración el significado del texto (semántica) a la hora de inferir, pero son muy costosos de configurar (problema del cuello de botella de la adquisición del conocimiento), y no se adaptan bien a diferentes lenguajes y dominios; y los métodos basados en redes neuronales (*Deep neural network based methods*), que aunque requieren gran cantidad de recursos computacionales, son capaces de adaptarse bien a diferentes lenguajes y dominios mostrando ser el estado del arte en este ámbito.

De esta forma, aprovechando el auge de las técnicas de aprendizaje profundo surgen métodos basados en redes neuronales como GloVe [10] (del inglés *Global Vectors for Word Representation*) y Word2vec [11], dos modelos que aprenden de forma no supervisada a representar las palabras como vectores en un espacio D-dimensional que contenga información semántica de estas. A estos vectores se les denomina *word-embeddings*.

Los *word-embeddings* de las palabras “perro” y “gato” estarían a una distancia más cercana en ese espacio, mientras que “gato” y “luna” estarían más distanciadas. Además, como podemos ver en el ejemplo de la Figura 3, permiten realizar operaciones matemáticas entre ellos que corresponden a cambios semánticos.

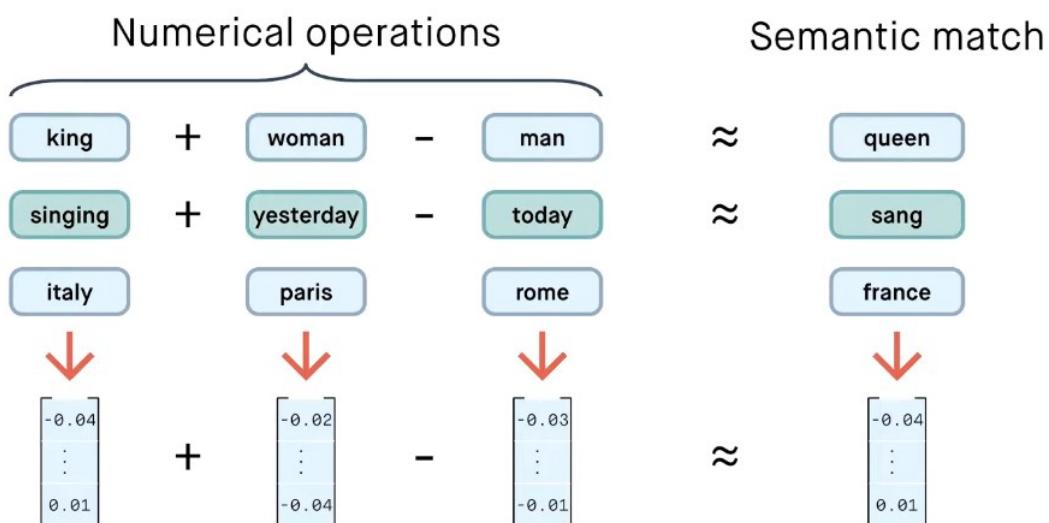


Figura 3. Ejemplo de operación sobre los word-embeddings.

Fuente: <https://www.youtube.com/watch?v=-9vVhYEXeyQ>

Sin embargo, aunque Word2vec y GloVe sean capaces de representar la información semántica de una palabra, esta depende del contexto en el que se encuentre en el texto y los *word-embeddings* generados con estos modelos son estáticos.

Es por eso por lo que aparece una nueva arquitectura basada en redes neuronales para solucionarlo: ELMo [12] (del inglés *Embeddings for Language Models*), capaz de enriquecer estos *word-embedding* de forma dinámica capturando la información contextual de esta. En la Figura 4 se puede ver cómo su arquitectura está formada por redes LSTM [13] (del inglés *Long Short-Term Memory*).

Estas últimas son una ampliación de las Redes Neuronales Recurrentes [13] (RNN, del inglés *Recurrent Neural Network*), cuya función es ampliar la memoria de estas y poder así recordar una mayor cantidad de información útil con la lectura de la entrada, ya que esta es procesada secuencialmente y, al avanzar en dicha lectura, la información que se tiene sobre los elementos que va dejando atrás es muy escasa.

De esta manera, ELMo realiza una concatenación de LSTM que procesa la entrada de izquierda a derecha y otras que lo hacen de derecha a izquierda, consiguiendo así una pseudo bidireccionalidad a la hora de calcular la representación enriquecida con el contexto de una palabra.

Sin embargo, esto supone un problema, ya que al procesarla de manera secuencial y hacia los dos sentidos el modelo generado es muy lento de entrenar.

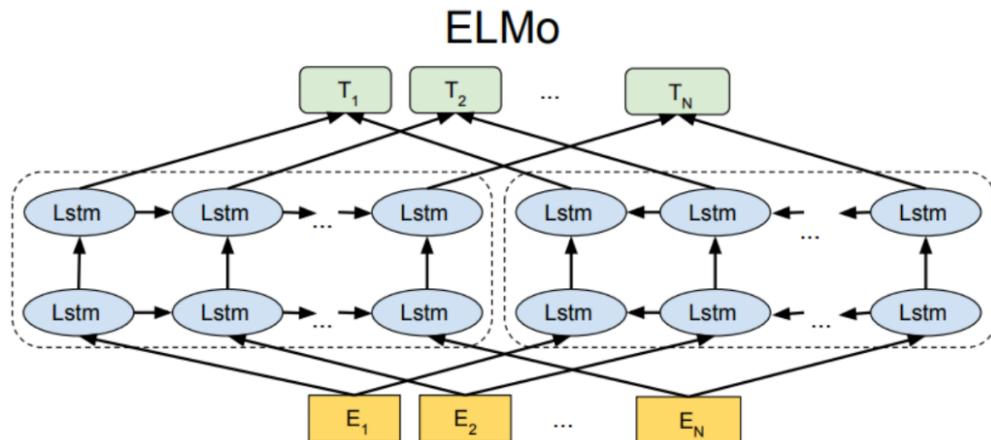


Figura 4. Arquitectura de ELMo.

Fuente: <https://programmerclick.com/article/2069798831/>

2.2. *Transformers*

A finales de 2017, Google revolucionó el campo del Aprendizaje Profundo con su publicación “*Attention is all you need*” [1], en el que presentaban una nueva arquitectura que haría historia: el *Transformer*.

Esta arquitectura nació con el objetivo de traducir entre el inglés y el francés o el inglés y el alemán, pero, aprovechando las ideas que se presentaban en dicha publicación, han ido apareciendo nuevas arquitecturas ampliando la familia de los *Transformers* y obteniendo nuevos avances en los diferentes ámbitos del Aprendizaje Profundo. Es por esto por lo que estudiaremos esta arquitectura, ya que es la base de los modelos que se utilizarán en este proyecto.

El principal problema que presentaban los predecesores del *Transformer* era la lentitud en la fase de entrenamiento y su “falta de memoria”, pues tenían un mal funcionamiento con secuencias largas debido a que a la hora de calcular un *word-embedding* de una palabra, esta solo podía enriquecerse con el contexto que le aporten palabras no muy distanciadas en la secuencia.

Como se ha podido observar hasta ahora, las palabras de la entrada eran procesadas en un orden dado y de una en una. Sin embargo, con la llegada de los *Transformers*, todas las palabras de la secuencia pasan a procesarse a la vez, resolviendo el problema de la paralelización y aprovechando de manera eficiente los avances del hardware, y más concretamente, la potencia de cómputo en GPU (del inglés *Graphics Processing Unit*) y TPU (del inglés *Tensor Processing Unit*).

Al procesar todas las palabras en paralelo, el *Transformer* introduce una verdadera bidireccionalidad y soluciona el problema de la mala memoria que presentaban sus predecesoras, ya que es capaz de entender mejor el significado de cada palabra al contar con información sobre el contexto completo en el que se encuentra.

2.2.1. Arquitectura

La arquitectura original del *Transformer*, a muy alto nivel, se puede ver como una caja negra que recibe una secuencia de entrada y produce otra de salida: la traducción de un texto

Más en detalle, está formada por dos grandes componentes: un conjunto de codificadores que procesan la secuencia de entrada y producen una representación vectorial de esta; y un conjunto de decodificadores que, usando dicha información, producen la secuencia de salida.

Podemos ver una ilustración de ello en la Figura 5, en la que la secuencia de entrada es una secuencia de palabras en francés y la secuencia de salida en una secuencia de palabras en inglés.

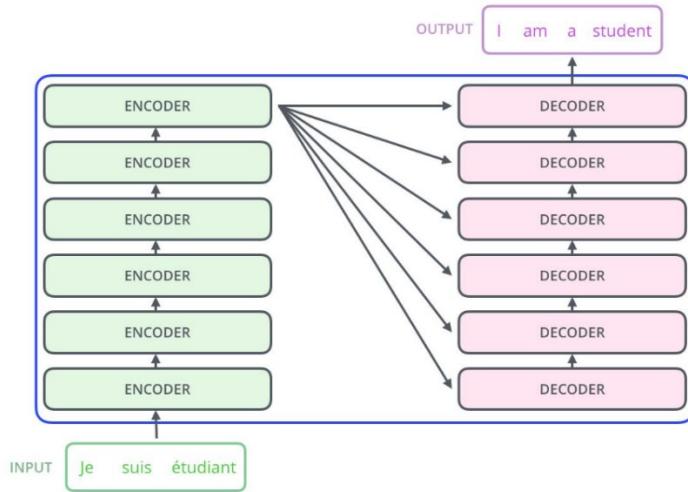


Figura 5. Arquitectura del Transformer a alto nivel.

Fuente: <https://programmerclick.com/article/30221772811/>

Apilar capas es lo que hace a una arquitectura de Aprendizaje Profundo potente y apilar codificadores y decodificadores es lo que permite que este modelo sea capaz de capturar la complejidad de un lenguaje completo. En la Figura 6 se muestra la arquitectura completa del *Transformer* en la que podemos ver que se divide en dos bloques: a la izquierda los codificadores y a la derecha los decodificadores como se había ilustrado en la Figura 5.

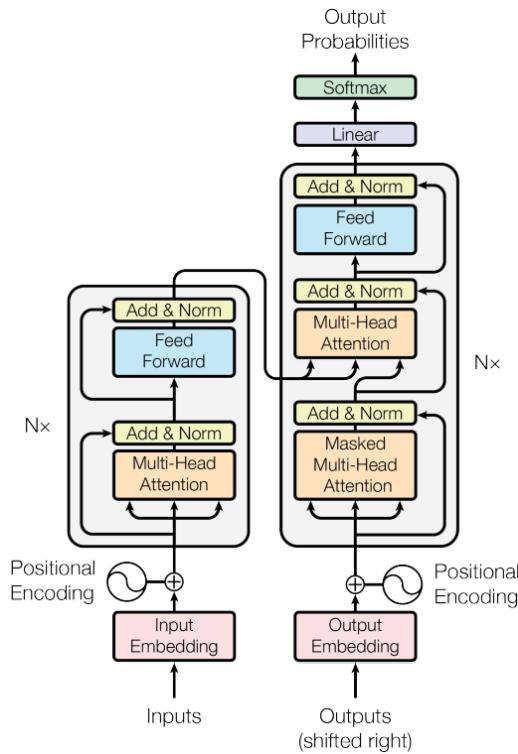


Figura 6. Arquitectura original del Transformer.

Fuente: <https://dl.acm.org/doi/10.5555/3295222.3295349>

El bloque de N codificadores recibe unos *word-embeddings* estáticos, es decir, que una misma palabra en diferentes contextos tienen la misma representación vectorial o *word-embedding*, y se va a encargar de enriquecerlos con la información del contexto que le aporte cada palabra en la secuencia. Para ello, este bloque adquiere un conocimiento del lenguaje durante el entrenamiento como se verá más adelante, que le hace capaz de aprender las relaciones entre las palabras y cómo afectan a su significado en la frase.

Como se ha mencionado anteriormente, los *Transformers* son capaces de realizar de forma paralela el procesamiento de los *word-embeddings* de una secuencia. Esto se refleja en cada codificador que, gracias a sus componentes, es capaz de recibir la secuencia completa de *word-embeddings* y enriquecer todos ellos con información del contexto simultáneamente.

Mientras que el bloque de codificadores solo necesita ejecutarse una vez para obtener las representaciones vectoriales de la secuencia de entrada, el bloque de decodificadores produce una palabra por cada iteración del bloque hasta llegar al final de la secuencia. Este bloque también adquiere conocimiento del lenguaje, ya que está aprendiendo la relación entre las palabras de diferentes idiomas gracias a sus representaciones vectoriales como se verá más adelante. En la Figura 7 se puede ver ilustrado un ejemplo del estado del *Transformer* durante una traducción de inglés a francés en el momento en el que se predice la segunda palabra de la secuencia de salida.

A continuación, se explicarán más en detalle los componentes que forman los codificadores y decodificadores, además de otros necesarios para hacer posible su funcionamiento.

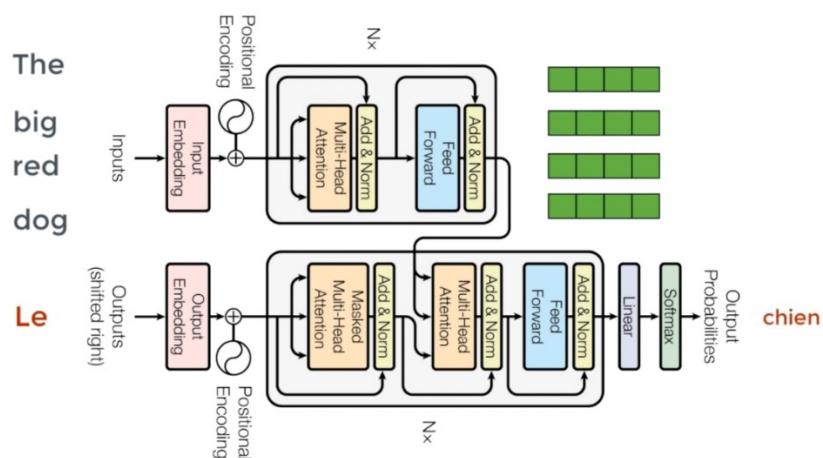


Figura 7. Ejemplo de traducción durante la predicción de la segunda palabra de la frase en inglés.

Fuente: <https://www.youtube.com/watch?v=TQQlZhbC5ps>

2.2.1.1. Codificación posicional

Este componente es uno de los más importantes junto al mecanismo de atención que se verá más abajo. Como se ha mencionado, las RNN y LSTM tienen inherentemente en cuenta el orden de las palabras, pues se procesan en secuencia. Sin embargo, con la llegada de los *Transformers*, todas las palabras se procesan en paralelo, perdiendo la información de la posición de cada palabra dentro de una oración. Por ello, surge la necesidad de un componente como este que sea capaz de aportar esta información que perdemos; por ejemplo, no es lo mismo decir “un pobre hombre” que “un hombre pobre” ni “el Barça derrota al Madrid” que “el Madrid derrota al Barça”. Para resolver este inconveniente implementaron una codificación posicional [14], en la que a cada *word-embedding* de la secuencia se le suma un vector de la misma dimensión que este y calculando cada posición de la siguiente manera:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Donde:

- *pos*: indica la posición de la palabra en la secuencia para la que estamos calculando el vector de codificación posicional.
- *i*: indica la dimensión del vector de codificación posicional para la que se está calculando el valor
- *d_{model}*: es la dimensión del modelo, o lo que es lo mismo, la dimensión de cada *word-embedding*.

Esta forma de realizar una codificación no es la única posible, pero con ella se consigue añadir información posicional única para cada palabra, determinista y con valores entre -1 y 1 que favorecen los cálculos del gradiente, lo cual le permite trabajar tanto con secuencias cortas como con extensas.

En la Figura 8 se puede observar un ejemplo gráfico de estos vectores, donde el eje vertical representa la posición del elemento en la secuencia procesada, y el eje horizontal la dimensión del *word-embedding*, de forma que a cada *word-embedding* original se le sume el vector correspondiente obtenido con este método.

Una vez los *word-embeddings* han pasado por este componente, han sido enriquecidos con la información de su posición en la secuencia.

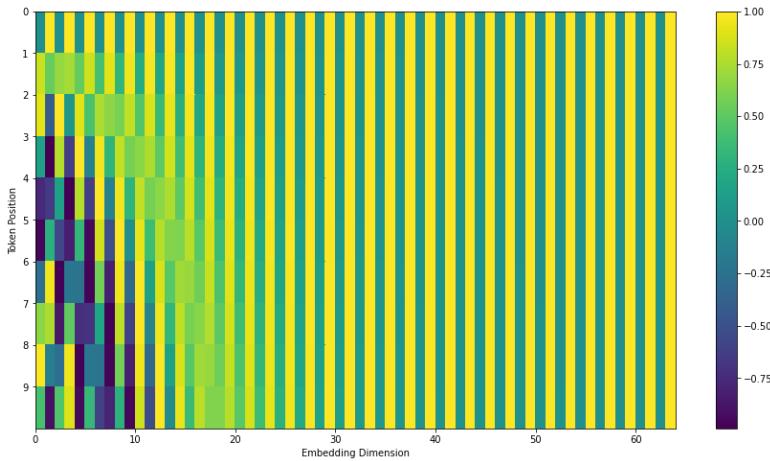


Figura 8. Ejemplo de codificación posicional.

Fuente: <https://jalammar.github.io/illustrated-transformer/>

2.2.1.2. Mecanismo de atención

A grandes rasgos, este mecanismo se encarga de enriquecer los *word-embeddings* con información del contexto. Por ejemplo, como se puede ver en la Figura 9, a la hora de aportar contexto al *word-embedding* de “it”, el mecanismo de atención calcula para la primera frase que la palabra más importante es “animal” mientras que en la segunda es “street” sin obviar el resto de las palabras pintadas en azul que también aportan contexto, pero en menor medida. En este ejemplo se calcula sobre una frase consigo misma, por lo que se denomina *self-attention*.

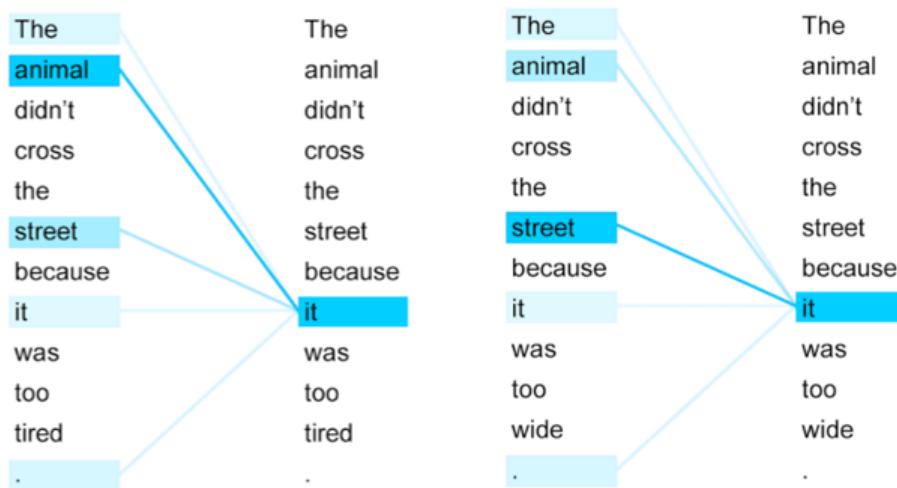


Figura 9. Representación gráfica del resultado de aplicar self-attention sobre dos ejemplos.

Fuente: <https://medium.com/iotforall/machines-learn-to-translate-sentences-more-naturally-bc6c2267769d>

Cuando este proceso se repite para cada uno de los *word-embeddings* de las palabras que forman la sentencia, obtenemos una matriz numérica indicando la compatibilidad entre palabras conocida como *matriz de atención*.

Este mecanismo también se puede aplicar entre dos secuencias diferentes. En el ejemplo ilustrado en la Figura 10, podemos ver la matriz de atención calculada por este mecanismo en un *Transformer* durante la traducción, en la que un color más claro representa un mayor grado de compatibilidad entre palabras. Se puede observar cómo es capaz de comprender la relación entre palabras, aunque estas estén en distinto orden dentro de la oración como es el caso de “*Area*” y “*zone*”.

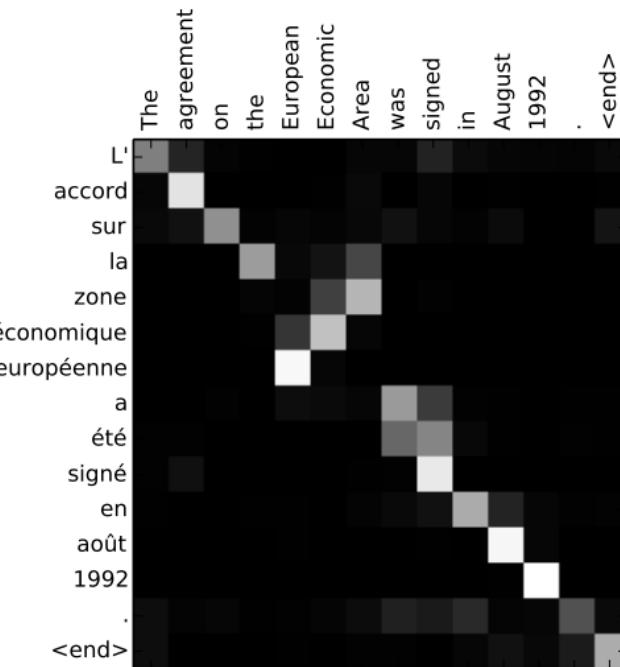


Figura 10. Matriz de atención.

Fuente: https://www.tensorflow.org/text/tutorials/nmt_with_attention

Los mecanismos de atención se utilizan en las redes neuronales recurrentes para solucionar el problema de memoria desde 2015 [15], pero, en el artículo de los *Transformers*, se describe cómo es posible utilizar solo esta combinación de mecanismos de atención y eliminar las redes recurrentes permitiendo la paralelización.

Entrando más en detalle, la función de atención propuesta en la publicación “*Attention is all you need*” tiene por nombre *Scaled Dot-Product Attention* y se puede ver ilustrada en la Figura 11.

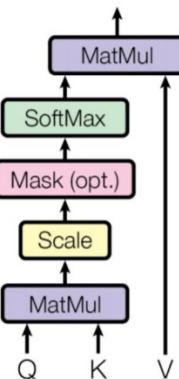


Figura 11. Función de atención “Scaled Dot-Product Attention”.

Fuente: <https://dl.acm.org/doi/abs/10.5555/3295222.3295349>

Esta función recibe tres matrices de entrada: consultas, claves y valores, siendo Q, K y V en la Figura 11 respectivamente. Estas tres matrices son proyecciones lineales de los vectores que forman la matriz de *word-embeddings* del texto y, tras pasar por el mecanismo de atención, este devuelve una nueva matriz de *word-embeddings* ya enriquecidos con contexto. Como se ha comentado, los *Transformers* destacan por su alta paralelización, que se puede ver reflejado en el cálculo con los vectores agrupados en matrices en vez de con los vectores de forma individual, permitiendo obtener el *word-embedding* de cada palabra enriquecido con el contexto de forma simultánea.

La idea de esta función calcular cuánto de relacionado está cada *word-embedding* con los demás de la secuencia haciendo el producto escalar del vector consulta de este con todos los vectores clave de la secuencia y, posteriormente, usar esta información para calcular una suma ponderada de los vectores valor y producir los nuevos *word-embeddings* como salida de la función.

Para calcular cuánto de relacionados están los *word-embeddings* y saber en qué proporción seleccionar cada vector *value* a la hora de calcular los nuevos *word-embeddings*, además de hacer la multiplicación de Q y K como se ha mencionado, se aplica un escalado sobre los datos y posteriormente una función de activación *softmax* que convierte cada vector de atención de la matriz formado por números reales en vectores con números reales en el rango [0,1] que suman 1.

En la Figura 12 se puede ver una ilustración de esta función de atención.

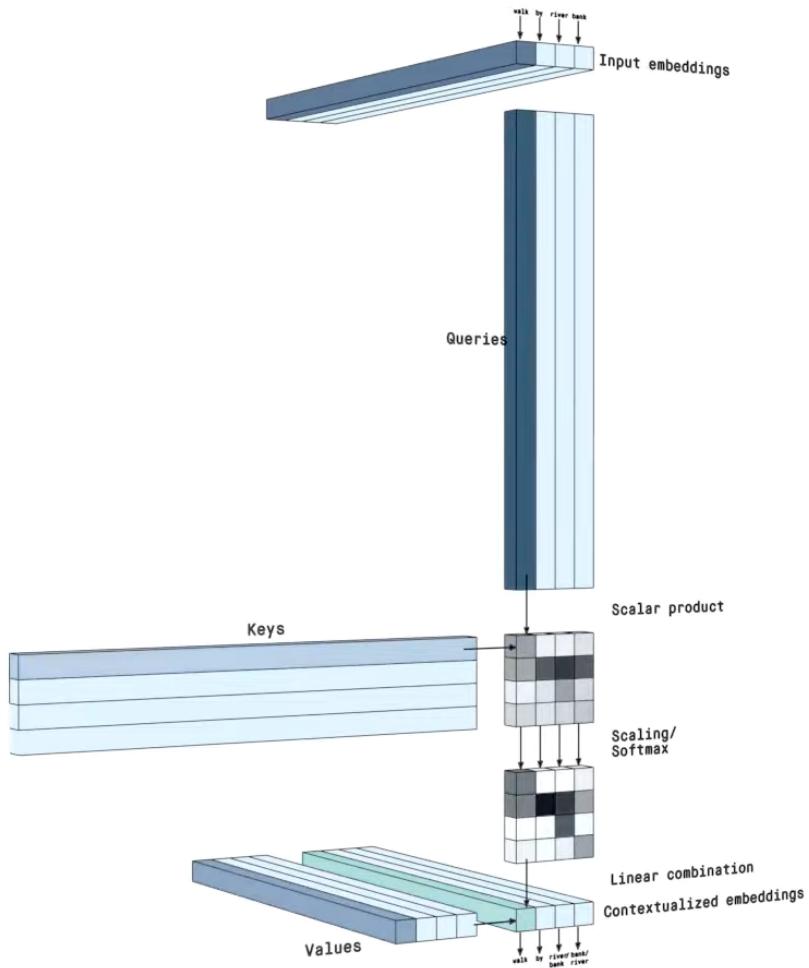


Figura 12. Representación de la función “Scaled Dot-Product Attention”.

Fuente: <https://www.youtube.com/watch?v=-9vVhYEXeyQ>

2.2.1.3. Multi-Head Attention

Una vez introducido el mecanismo de atención, se verá que en la práctica existen varios mecanismos de atención que funcionan en paralelo llamados *heads* o cabezas.

De esta manera, en vez de aplicar un solo mecanismo de atención sobre los vectores clave, consulta, y valor con sus dimensiones originales, para cada cabeza se realizan diferentes proyecciones lineales (inicializadas aleatoriamente y aprendidas durante el entrenamiento) de cada uno de estos vectores obteniendo unos nuevos de menor dimensionalidad. Debido a esta reducción y ya que se calculan en paralelo, el coste computacional es el mismo que aplicar una sola cabeza resultando este mucho más eficaz.

Una vez obtenidas las matrices valor de las cabezas, son concatenadas y proyectadas linealmente otra vez multiplicándolas por otras matrices aprendidas previamente durante el entrenamiento.

En la Figura 13 se puede ver una ilustración del mecanismo de *Multi-Head Attention* que, como se puede observar, es muy parecido al mecanismo de atención visto en la Figura 12.

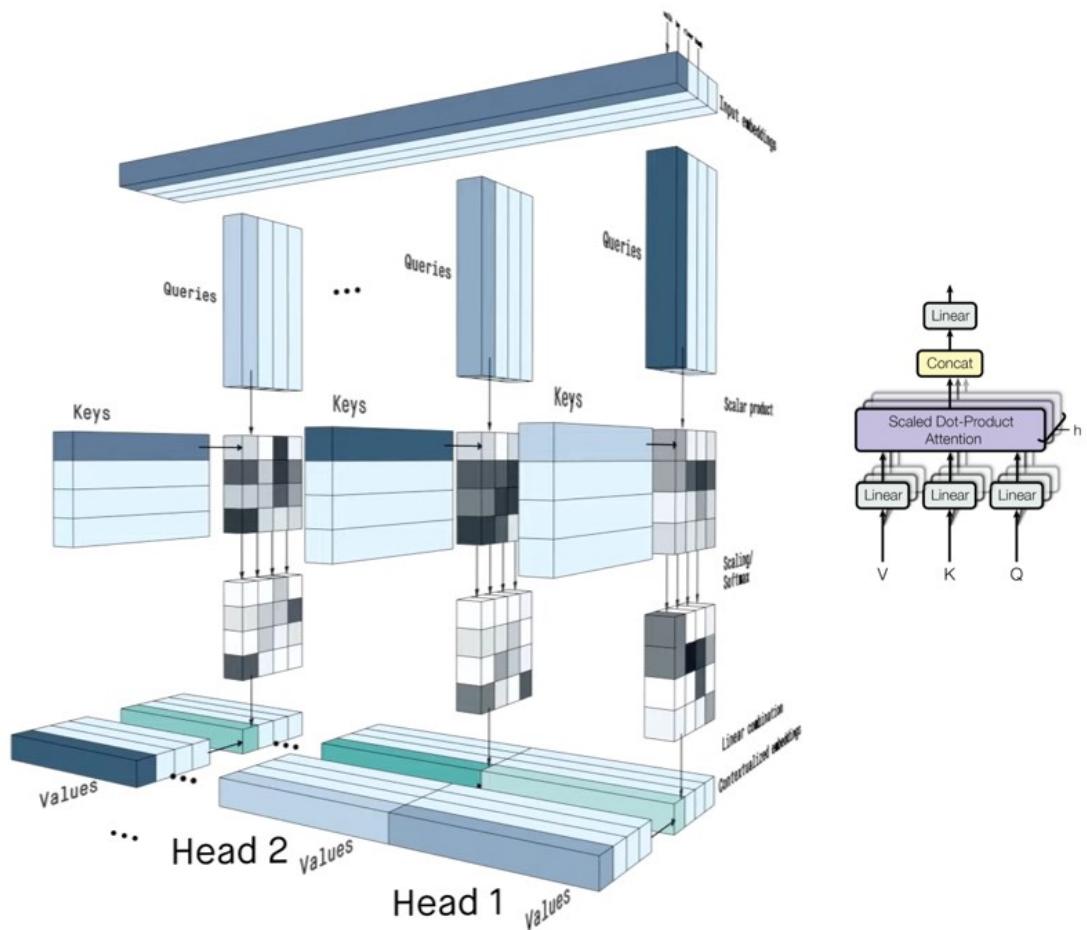


Figura 13. Mecanismo de Multi-Head Attention.

Fuente: <https://www.youtube.com/watch?v=-9vVhYEXeyQ>

Como se puede ver en la arquitectura original de los *Transformers* en la Figura 6, este mecanismo se aplica de tres formas diferentes:

- Atención en el codificador: se aplica un mecanismo de *self-attention*, en el que Q, V y K se obtienen de la salida del codificador anterior en el bloque. De esta manera es capaz de calcular la relación entre las palabras de una misma frase. Es el que se aplica en el ejemplo de la Figura 9.

- Atención en el decodificador: se aplica un mecanismo de *self-attention* ligeramente modificado llamado *Masked Multi-Head Attention*, en el que Q, V y K se obtienen de la salida del decodificador anterior en el bloque y tiene como objetivo evitar que el decodificador haga “trampas” en el entrenamiento aplicando una máscara a las palabras que tiene por delante en cada paso de la traducción, de modo que solo pueda utilizar la información que ha obtenido hasta el momento.
- Atención en codificador-decodificador: en el decodificador se puede encontrar otra capa más de atención, en la que las claves y los valores se obtienen de la salida del bloque de los codificadores mientras que las consultas se obtienen de la capa anterior en el decodificador, permitiendo así poder atender a todas las palabras de la secuencia de entrada y las que lleva generadas. De esta manera el bloque de decodificadores es capaz de aprender la relación entre palabras en distinto idioma durante el entrenamiento.

2.2.1.4. Normalización

Esta capa tiene como objetivo normalizar los vectores para que el entrenamiento sea más rápido y para evitar que los valores se vuelvan demasiado grandes de modo que se mantengan dentro de un rango más pequeño y manejable.

A la hora de normalizar existen dos métodos diferentes: *batch normalization* y *layer normalization*. Como se puede observar en la Figura 14, el primero calcula los coeficientes y realiza la normalización en base a cada ejemplar del conjunto, mientras que el segundo lo hace en base a cada dimensión. Ambas normalizaciones se basan en obtener un promedio cero y una varianza cercana al uno, aunque los desarrolladores se decantaron por el segundo método ya que su rendimiento en tareas del NLP es superior.

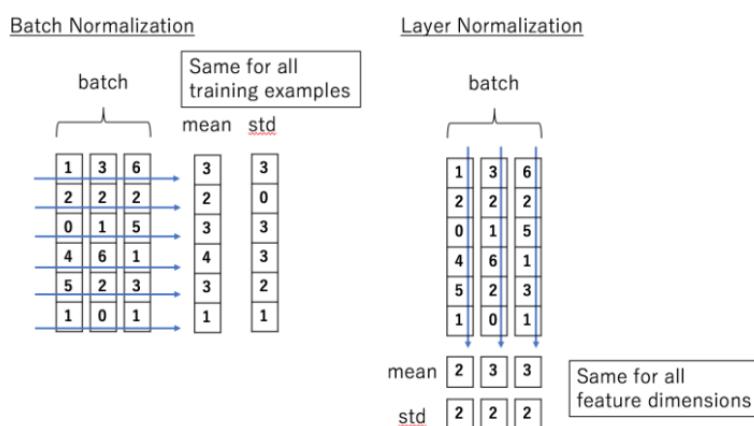


Figura 14. Batch Normalization vs. Layer Normalization.

Fuente: <https://chowdera.com/2022/03/202203180452132413.html>

2.2.1.5. Red neuronal densa

Este componente es una red neuronal densa o *Fully-Connected Feed-Forward Neural Network* [16] que tiene como propósito hacer una proyección lineal de la salida de una capa de atención para obtener una representación más rica y, seguidamente, servir de entrada para la siguiente capa.

Esta red neuronal es aplicada a cada *word-embedding* de manera independiente e idénticamente ya que los parámetros aprendidos son los mismos para la misma secuencia permitiendo que se puedan realizar inferencias en paralelo.

Entrando más en detalle sobre esta red, en la publicación la definen de la siguiente manera:

$$FNN(x) = \max(0, xW_1 + b_1) W_2 + b_2$$

Donde:

- La entrada de la red neuronal x es el vector resultado de la capa de normalización anterior como podíamos ver en la arquitectura del *Transformer* en la Figura 6.
- W_1 , W_2 , b_1 , y b_2 son los parámetros aprendidos por la red durante el entrenamiento (pesos y sesgos de las dos capas).
- La función $\max(0, xW_1 + b_1)$ es una función de activación conocida como rectificador o ReLu [17].

En la publicación “*Transformer Feed-Forward Layers Are Key-Value Memories*” [18], se realiza un estudio sobre lo que aprenden estas redes durante el entrenamiento y descubren cómo han aprendido a detectar diferentes patrones semánticos. Más concretamente, describen cómo en las primeras capas de los bloques estas redes captan patrones más superficiales, mientras que en las últimas capas detectan patrones semánticos. Esto se puede ver ilustrado en la Figura 15, donde el número de codificadores y decodificadores es 16.

En la Figura 16 se pueden observar ejemplos de patrones que han aprendido a detectar estas redes neuronales, donde la columna *Key* identifica estos patrones. El superíndice indica la capa en la que se encuentra de las 16 existentes en el ejemplo, mientras que el subíndice indica que patrón de esa capa es, que en este caso asciende a 4096 posibles elementos. Esto produce un total de 65.536 *Keys* o patrones posibles.

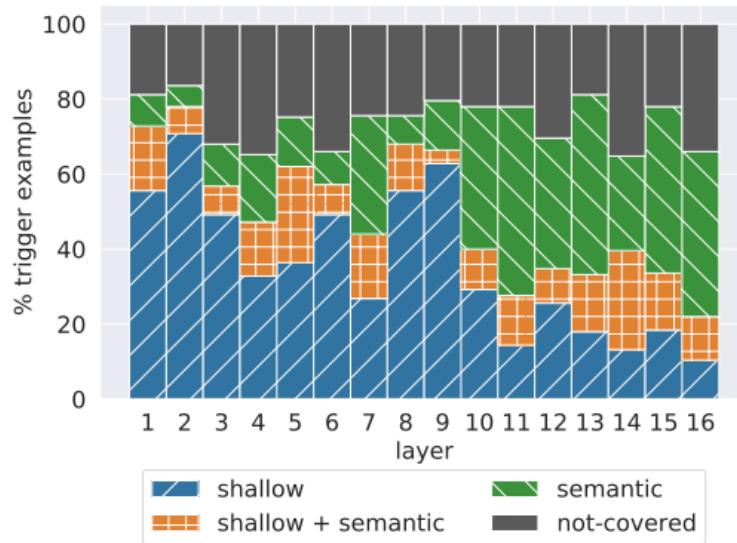


Figura 15. Representación del tipo de patrones detectados en diferentes capas de un Transformer.

Fuente: <https://aclanthology.org/2021.emnlp-main.446/>

Key	Pattern	Example trigger prefixes
k_{449}^1	Ends with “substitutes” (shallow)	<i>At the meeting, Elton said that “for artistic reasons there could be no substitutes In German service, they were used as substitutes Two weeks later, he came off the substitutes</i>
k_{2546}^6	Military, ends with “base”/“bases” (shallow + semantic)	<i>On 1 April the SRSG authorised the SADF to leave their bases Aircraft from all four carriers attacked the Australian base Bombers flying missions to Rabaul and other Japanese bases</i>
k_{2997}^{10}	a “part of” relation (semantic)	<i>In June 2012 she was named as one of the team that competed He was also a part of the Indian delegation Toy Story is also among the top ten in the BFI list of the 50 films you should</i>
k_{2989}^{13}	Ends with a time range (semantic)	<i>Worldwide, most tornadoes occur in the late afternoon, between 3 pm and 7 Weekend tolls are in effect from 7:00 pm Friday until The building is open to the public seven days a week, from 11:00 am to</i>
k_{1935}^{16}	TV shows (semantic)	<i>Time shifting viewing added 57 percent to the episode’s The first season set that the episode was included in was as part of the From the original NBC daytime version , archived</i>

Figura 16. Ejemplos de patrones que capturan diferentes keys en el estudio.

Fuente: <https://aclanthology.org/2021.emnlp-main.446/>

2.2.2. Resultados

Los resultados que obtienen los *Transformers* a la hora de traducir han resultado ser el estado del arte. Desde el momento de su aparición superó con creces a los predecesores. Como se puede ver en la Figura 18, esta arquitectura obtiene los mejores resultados en el algoritmo *BLEU* [19] (del inglés *bilingual evaluation understudy*) que evalúa la calidad del texto traducido por una máquina de un lenguaje a otro dándole una puntuación.

Los modelos *Transformer* evaluados son *Transformer big* y *Transformer base* presentados en la publicación original de los *Transformers* y cuyas configuraciones pueden verse ilustrados en la Figura 17, donde:

- N : es el número de codificadores/decodificadores.
- D_{model} : dimensión de los *word-embeddings* producidos.
- D_{ff} : número de neuronas en la capa oculta de las redes neuronales densas.
- h : número de cabezas del mecanismo de *Multi-head Attention*.
- D_k : dimensión de los vectores *Key*, es decir, las proyecciones lineales de cada *word-embedding* durante el mecanismo de *Multi-head Attention*.
- D_v : dimensión de los vectores *Value*.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps
base	6	512	2048	8	64	64	0.1	0.1	100K
big	6	1024	4096	16			0.3		300K

Figura 17. Parámetros de *Transformers base* y *Transformers big*.

Fuente: <https://dl.acm.org/doi/abs/10.5555/3295222.3295349>

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.0		$2.3 \cdot 10^{19}$

Figura 18. Puntuación de diferentes modelos con el algoritmo *BLEU*.

Fuente: <https://dl.acm.org/doi/abs/10.5555/3295222.3295349>

Habiendo comprendido el trasfondo de los *Transformers* y sus componentes puede verse como los componentes por separado se pueden aprovechar para formar diferentes redes neuronales:

- Si se utiliza solamente el bloque de decodificadores, obtenemos modelos como XLNet [20], y GPT y sus versiones modernas, que permiten la generación de texto.
- Si se utiliza solamente el bloque de codificadores obtenemos modelos capaces de generar representaciones precisas y ricas en información semántica como, por ejemplo, BERT que será el que veremos a continuación.

2.2.3. *Transformers*: BERT

En 2018 aparece una nueva publicación de Google: “*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*” [21] que, usando parte de la arquitectura de los *Transformers* y, más en concreto, el bloque de codificadores consigue una red neuronal capaz de resolver diferentes tareas, como por ejemplo:

- Responder preguntas sobre un párrafo.
- Analizar sentimientos de un texto.
- Resumir textos.
- Clasificación de documentos.
- Completar palabras en un texto.
- Extracción de entidades o *Name Entity Recognition* (NER).
- Determinar si dos textos afirman lo mismo, se contradicen o son neutrales. *Natural Language Inference* (NLI)

En la Figura 19 se puede observar una ilustración de la arquitectura de BERT para alguna de las anteriores tareas mencionadas. Como veremos, no hace falta hacer cambios en la arquitectura interna de BERT para conseguir adaptar el modelo a diferentes tipos de tareas.

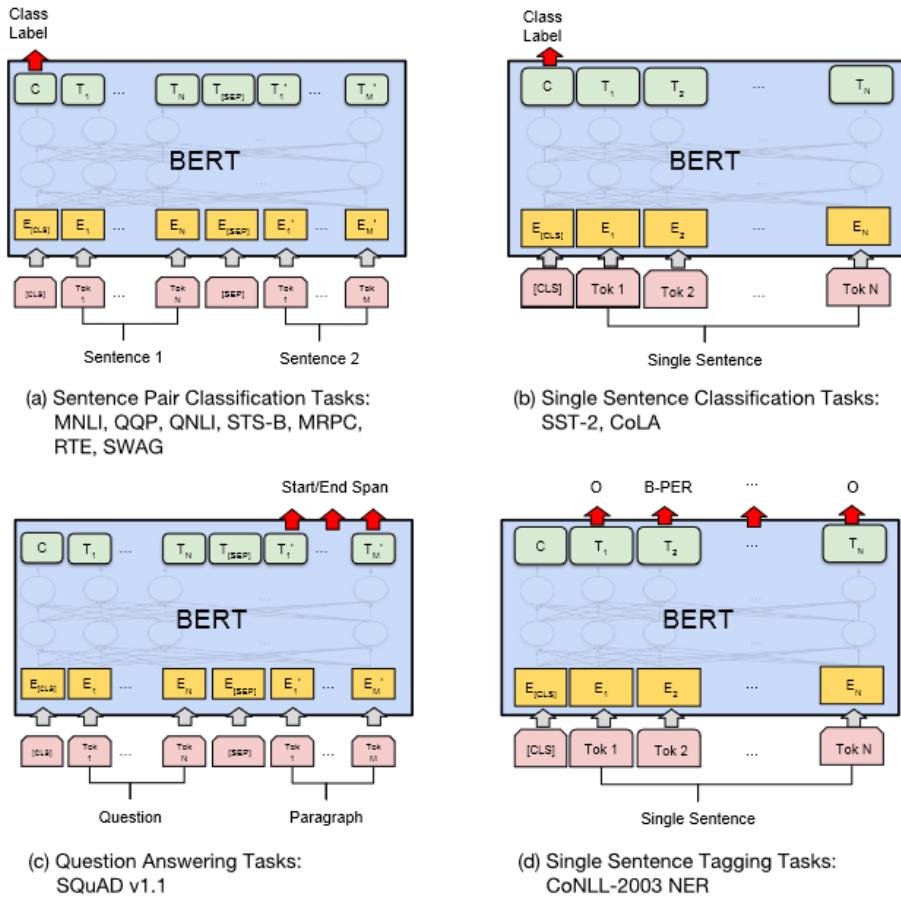


Figura 19. Diferentes tipos de tareas que BERT puede realizar.

Fuente: <https://aclanthology.org/N19-1423/>

BERT es una arquitectura bidireccional al igual que los *Transformers*, lo que quiere decir que analiza las palabras de una frase tanto por la izquierda como por la derecha, es decir, relaciona las palabras entre sí en vez de considerarlas de manera individual. Esto es así ya que su arquitectura usa la idea de los codificadores de los *Transformer*, que usando el mecanismo de atención (*Multi-Head Attention*) permite obtener *word-embeddings* ricos en contexto y así proporcionar una representación vectorial más adecuada.

En la Figura 20 se puede observar la arquitectura de BERT y se puede ver cómo utiliza 12 capas de codificadores en la que cada una utiliza un mecanismo de *Multi-Head Attention* como el que se ha ilustrado en la Figura 13.

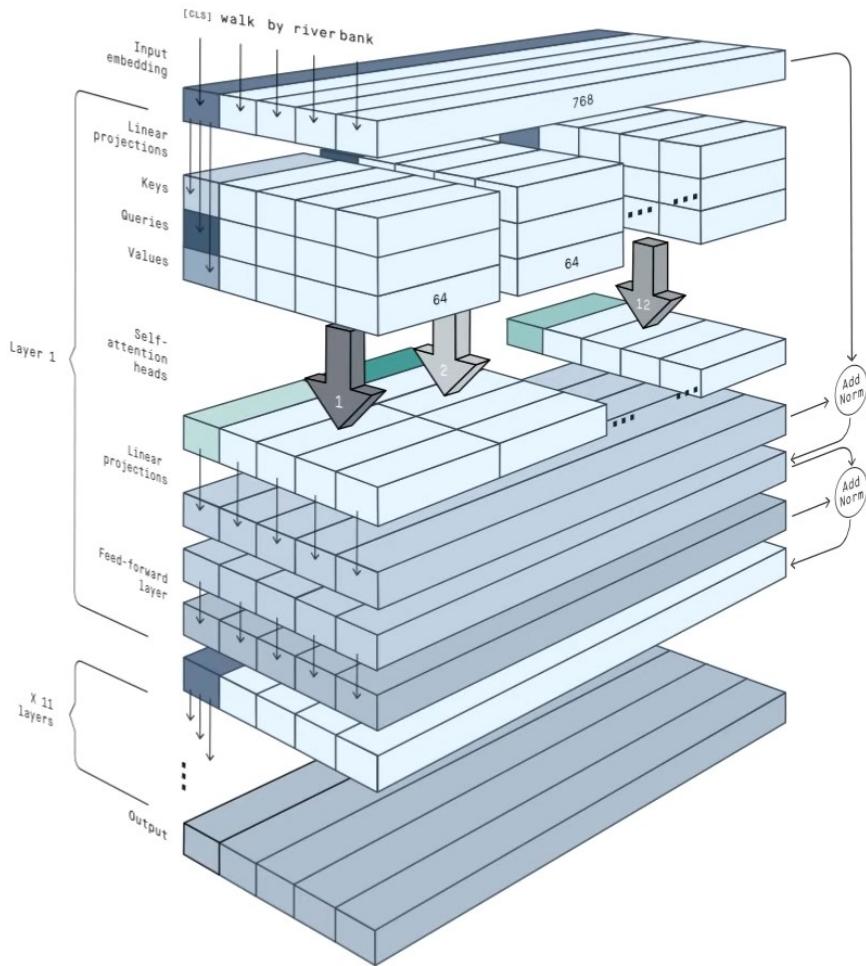


Figura 20. Arquitectura interna de BERT.

Fuente: <https://www.youtube.com/watch?v=-9vVhYExeyQ>

Para conseguir este modelo primero hay que enseñarle de alguna manera *qué es el lenguaje* y, una vez BERT tenga conocimiento sobre este, podemos entrenarlo para que resuelva una tarea en específico. Estos procesos se denominan respectivamente preentrenamiento y ajuste fino, los cuales se explicarán un poco más en detalle en esta sección.

En el caso de BERT, se ha entrenado con más de 2700 millones de palabras procedentes de Wikipedia¹ y 800 millones de palabras procedentes de libros y se ha aplicado en el buscador de Google.

¹ <https://es.wikipedia.org/wiki/Wikipedia:Portada>

2.2.3.1. Preentrenamiento

En esta fase, BERT adquiere el conocimiento del lenguaje y del contexto. Se consigue mediante dos procesos no supervisados de entrenamiento (es decir, no necesitan las respuestas correctas si no que las infieren de los datos) que se realizan simultáneamente:

- **MLM [21] (del inglés *Masked Language Models*)**: este método consiste en aplicar una máscara a palabras de los textos de entrenamiento de manera que estén ocultas para el modelo e intente predecir cuáles son estas infiriendo una distribución de probabilidad con las posibles palabras del vocabulario que encajan. Si se aplica la máscara a muchas palabras no habría suficiente contexto y, si se aplica a pocas, sería muy costoso de entrenar por lo que los desarrolladores eligieron seleccionar las palabras a las que aplicar la máscara en base al siguiente método:
 - Se selecciona el 15% de las palabras de cada texto como posibles palabras a ocultar.
 - De las palabras seleccionadas, el 80% de las veces se ocultan, el 10% se remplazan por una palabra aleatoria, y el otro 10% de las veces no se modifica.
- **NSP [21] (del inglés *Next Sentence Prediction*)**: este método consiste en proporcionarle dos frases a BERT y este tiene que aprender si una frase sigue a otra para poder comprender el contexto entre sentencias y ver cómo se relacionan. Más específicamente, funciona eligiendo pares de sentencias de los datos de entrenamiento de forma que el 50 % de las veces la segunda sentencia del par realmente es la siguiente de la primera y el otro 50% la segunda sentencia es una frase aleatoria del conjunto de entrenamiento.

Una vez usadas las dos técnicas en conjunto BERT ha adquirido el conocimiento del lenguaje y se puede empezar a entrenar para resolver un tipo de tarea concreto, lo que se conoce como ajuste fino.

2.2.3.2. Ajuste fino

Una vez que BERT ha realizado el preentrenamiento y tiene conocimiento sobre el lenguaje, podemos entrenarlo para resolver una tarea específica. Para conseguirlo, necesitamos pasarle un pequeño conjunto de datos etiquetado esta vez para poder realizar un aprendizaje supervisado.

Por ejemplo, si queremos entrenar nuestro modelo para que sea capaz de resolver preguntas, debemos aportar un conjunto de datos con preguntas y sus respectivas respuestas valoradas, de modo que aprenda a responder a dichas preguntas.

De esta manera, se puede entrenar una arquitectura para adquirir el conocimiento de un lenguaje y luego realizar este proceso de ajuste para enseñarle tareas en concreto a realizar, evitando así tener que entrenar desde el principio un modelo para cada tarea. En la Figura 21 se pueden ver ilustradas las arquitecturas de estos procesos.

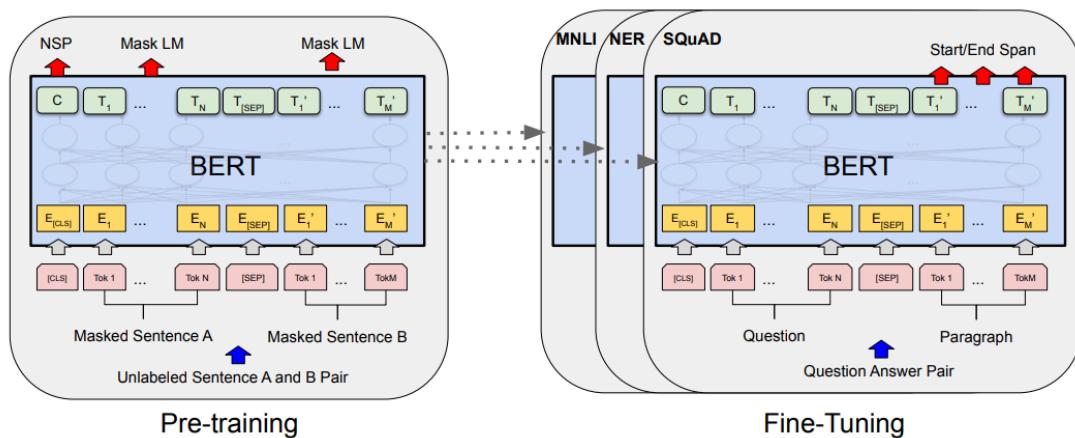


Figura 21. Esquema de preentrenamiento y ajuste fino de BERT.

Fuente: <https://aclanthology.org/N19-1423/>

2.2.3.3. Similitud semántica con BERT: *Cross-Encoders* y *Bi-Encoders*

Para poder medir la similitud semántica, se puede realizar un ajuste fino con un conjunto de pares de textos etiquetados con su similitud, como por ejemplo el conjunto STS *Benchmark dataset*¹, que contiene los pares de sentencias junto a su similitud en un rango de 0 a 5, que puede ser normalizado a un rango de 0-1 dividiendo la similitud entre 5.

Así, estos modelos que se basan en *Transformers*, una vez entrenados para medir la similitud semántica obtienen muy buenos resultados. A este tipo de modelos se le denominan *Cross-Encoders* o codificadores cruzados.

En estos modelos, las dos cadenas de textos a comparar son concatenadas y se introducen simultáneamente al modelo como se puede ver en la Figura 22, permitiendo al mecanismo de atención correlacionar también *word-embeddings* de ambas secuencias entre sí además de consigo mismas.

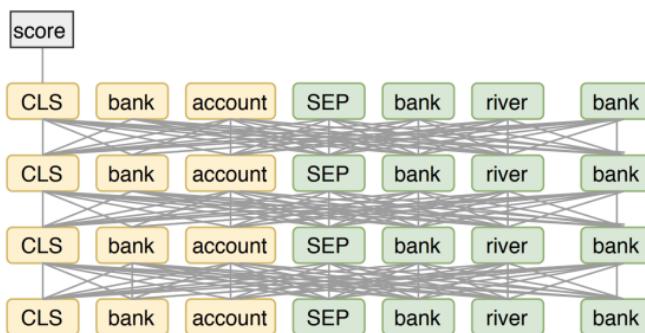


Figura 22. *Cross-Encoder* durante la inferencia de similitud.

Fuente: <https://www.amazon.science/blog/improving-unsupervised-sentence-pair-comparison>

Los *Cross-Encoders* basados en BERT obtiene muy buenos resultados, pero su complejidad computacional es bastante elevada y no es factible cuando se escala, ya que no produce *sentence-embeddings* en sí, si no que en cada inferencia calcula de nuevo los embeddings de ambas sentencias prestando atención a todos los elementos produciendo directamente la similitud. De esta forma, si se quiere encontrar el par más similar en un conjunto de n datos con BERT, es necesario que se le proporcione como entrada todos los pares posibles de sentencias en el conjunto, por lo que habría que usar BERT $\frac{n(n-1)}{2}$ veces. En una GPU moderna con un conjunto de datos de 10.000 elementos se necesitarían aproximadamente 50 millones de inferencias de BERT, y tardaría alrededor de 65 horas en ejecutarse.

¹ <https://deepai.org/dataset/sts-benchmark>

Por lo tanto, no resulta un enfoque adecuado para realizar una exploración semántica en estos casos.

Otro enfoque para medir la similitud semántica usando BERT es usar los *word-embeddings* de una secuencia producidos por los codificadores y obtener a partir de estos *word-embeddings* orientados a palabras, *embeddings* asociados a sentencias, a los que se les denominan *sentence-embeddings*.

La manera más común de obtener un *sentence-embedding*, es tomar los *word-embeddings* que producen internamente en la última capa de la arquitectura y usando una de las siguientes funciones de *pooling* [3] sobre ellos, para obtener un *sentence-embedding*:

- Usando el *embedding* CLS [21] que proporciona BERT en la salida para las tareas de clasificación y que contiene información sobre la secuencia completa.
- Realizando una media de cada dimensión de los word-embeddings de la secuencia. Esto se conoce como *mean-pooling*.
- Seleccionando el máximo de cada dimensión de los word-embeddings de la secuencia. Esto se conoce como *max-pooling*.

A este tipo de modelo se le denomina *Bi-Encoder* y lo podemos ver ilustrado junto al *Cross-Encoder* en la Figura 23.

Sin embargo, se ha visto que estos métodos directos para obtener *sentence-embeddings* no funcionan demasiado bien, siendo superados por *word-embeddings* generados por GLoVE y realizando la media en cada dimensión de estos.

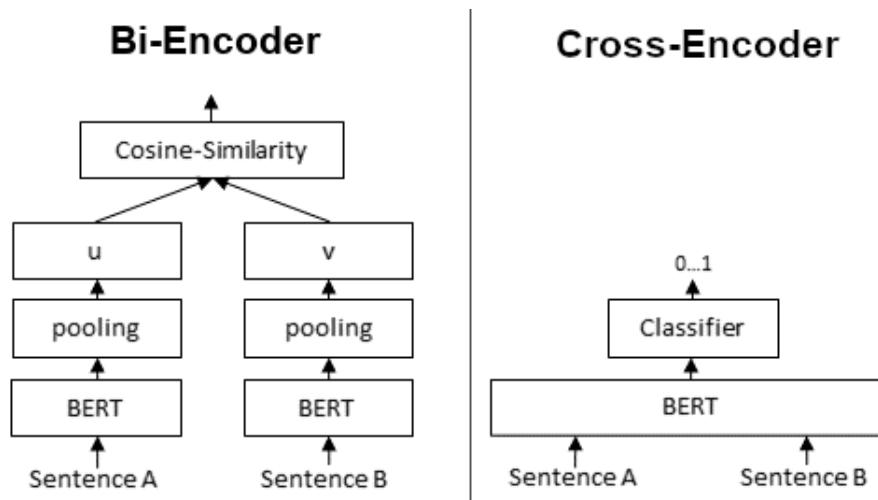


Figura 23. Bi-Encoder vs. Cross-Encoder.

Fuente: <https://www.sbert.net/examples/applications/cross-encoder/README.html>

Otro enfoque, más rápido a este y con mejores resultados, es usar modelos *Sentence-Transformers* para obtener los *Bi-Encoders*, que veremos en el siguiente punto, y que finalmente es el tipo arquitectura que se usará en este proyecto.

2.2.4. *Bi-Encoders* mediante *Sentence-Transformers*: SBERT

En 2019 se propone un nuevo modelo de *Bi-Encoder*, *Sentence-BERT* [3], que es resultado de un ajuste fino de BERT para conseguir que genere mejores *sentence-embeddings* en contenido semántico para que posteriormente puedan ser comparados mediante una medida de similitud como la similitud del coseno.

Este modelo es un *Bi-Encoder* como el que se había descrito en la Figura 23 en el apartado anterior, pero, a diferencia de este, se entrena mediante redes neuronales siamesas y tripletas [22] antes de la inferencia para mejorar los resultados que ofrecían los métodos directos descritos anteriormente.

De esta manera, según afirman en su publicación, para encontrar el par de sentencias más similares en un conjunto de datos, se puede calcular los *sentence-embeddings* de cada sentencia individual en 5 segundos, y luego calcular las similitudes del coseno entre los *sentence-embeddings* de cada par en apenas 0.01 segundos. Además, se obtienen mejores resultados que los anteriores modelos del estado del arte para la mayoría de las tareas de similitud semántica.

Entrando más en detalle sobre el entrenamiento con redes neuronales siamesas, la idea es ajustar los pesos de BERT teniendo un conjunto de datos que nos diga cuánto de cerca o de lejos deben de estar los *sentence-embeddings* que se producen.

El modelo de SBERT original realiza un ajuste fino con los conjuntos de datos Stanford Natural Language Inference (SNLI¹) y Multi-Genre NLI (MNLI²). Estos conjuntos contienen 570.000 y 430.00 pares de sentencias formados por una premisa y una hipótesis y etiquetadas de la siguiente manera:

- *Entailment*: la premisa sugiere la hipótesis.
- *Neutral*: la premisa y la hipótesis no están relacionadas.
- *Contradiction*: la premisa y la hipótesis se contradicen.

Para ello, se alimenta a BERT primero con la primera premisa obteniendo los *word-embeddings* y posteriormente se realiza con la hipótesis. Tras aplicar una capa de *Pooling* a cada conjunto de *word-embeddings*, obtenemos los dos *sentence-embeddings*, para finalmente formar un nuevo vector concatenando estos dos junto a su diferencia.

Este vector pasa por una FFNN (del inglés Feed Forward Neural Network) que produce tres salidas, una por cada posible clase y, que aplicando una función

¹ <https://nlp.stanford.edu/projects/snli/>

² <https://cims.nyu.edu/~sbowman/multinli/>

softmax, obtiene las probabilidades de pertenecer a cada clase. Esto junto a la etiqueta con la clase real del par, se usa para optimizar la función *cross-entropy loss* ajustando los pesos de BERT para que los *sentence-embeddings* de los pares de sentencias de la clase *Entailment* estén más cerca en el espacio, y los pares de la clase *Contradiction* más distantes.

En la Figura 24 podemos ver ilustrado a SBERT durante el entrenamiento y durante la inferencia. Es importante recordar que no se usan dos modelos BERT diferentes, si no que se usa uno solo que procesa en secuencia cada elemento del par.

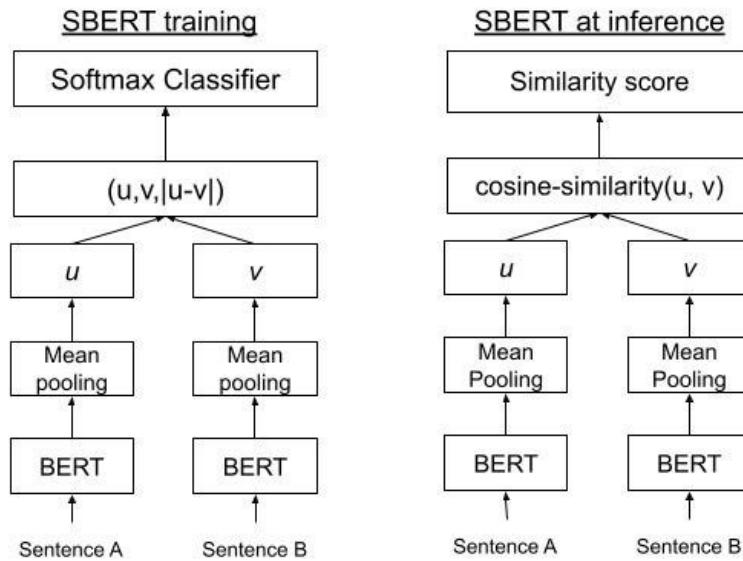


Figura 24. SBERT durante el entrenamiento y la inferencia

Fuente: <https://towardsdatascience.com/an-intuitive-explanation-of-sentence-bert-1984d144a868>

En la **¡Error! No se encuentra el origen de la referencia.** podemos ver el desempeño de estos modelos sobre el conjunto de datos de prueba de STS *Benchmark*. Se puede observar en los resultados que:

- Sin estar entrenado específicamente sobre ningún conjunto específico de entrenamiento STS *Benchmark*, solamente con el entrenamiento mencionado anteriormente supera a los métodos predecesores.
- Entrenado en el conjunto de entrenamiento de STS *Benchmark* mediante *Regression Objective Function* [3] que utiliza la función de error cuadrático medio para entrenar al modelo, tiene puntuaciones muy similares a los predecesores.

- Entrenado primero en varios conjuntos de NLI y luego en el conjunto de entrenamiento STS *Benchmark* supera a los predecesores, siendo esta la mejor opción.

Model	Spearman
<i>Not trained for STS</i>	
Avg. GloVe embeddings	58.02
Avg. BERT embeddings	46.35
InferSent - GloVe	68.03
Universal Sentence Encoder	74.92
SBERT-NLI-base	77.03
SBERT-NLI-large	79.23
<i>Trained on STS benchmark dataset</i>	
BERT-STSb-base	84.30 ± 0.76
SBERT-STSb-base	84.67 ± 0.19
SRoBERTa-STSb-base	84.92 ± 0.34
BERT-STSb-large	85.64 ± 0.81
SBERT-STSb-large	84.45 ± 0.43
SRoBERTa-STSb-large	85.02 ± 0.76
<i>Trained on NLI data + STS benchmark data</i>	
BERT-NLI-STSb-base	88.33 ± 0.19
SBERT-NLI-STSb-base	85.35 ± 0.17
SRoBERTa-NLI-STSb-base	84.79 ± 0.38
BERT-NLI-STSb-large	88.77 ± 0.46
SBERT-NLI-STSb-large	86.10 ± 0.13
SRoBERTa-NLI-STSb-large	86.15 ± 0.35

Figura 25. Tabla de rendimientos de diferentes modelos en generación de sentence-embeddings.

La puntuación se representa como $100 * \text{coeficiente de correlación de Spearman}$.

Fuente: <https://aclanthology.org/D19-1410/>

3. Arquitectura del proyecto

Este proyecto está compuesto por dos grandes bloques:

- Un cuaderno interactivo realizado en Google Colaboratory¹ que, dada una colección de datos multimedia, es capaz de procesar las descripciones textuales de los componentes mediante el uso de técnicas PLN basadas en *Transformers* con el fin de generar la información necesaria para realizar una exploración del conjunto de datos de manera sencilla e interactiva.
- Un navegador web diseñado en *React*² que permite navegar por una colección de películas basándose en la similitud de su sinopsis. Estas similitudes son generadas con ayuda del cuaderno interactivo mencionado.

En este punto se explica más a fondo la arquitectura y cada uno de los componentes que hacen posible su funcionamiento.

3.1. SBMC: *Similarity-based browser of multimedia content*

3.1.1. Introducción

Esta herramienta es un cuaderno interactivo desarrollado en el entorno Google Colaboratory que, dada una colección de datos, facilitará el análisis de ésta midiendo la similitud semántica de uno de los atributos. De esta manera se facilita la exploración la colección poniendo a disposición varias herramientas como se explica más adelante.

Una de las piezas fundamentales de este trabajo es el uso de los modelos *Sentence-Transformers*, que pueden ser descargados de la web de Hugging-Face³ y donde se puede encontrar multitud de estos modelos ya entrenados en diferentes conjuntos de datos. Podemos ver en la Figura 26 algunos de los modelos más descargados de la web para calcular la similitud semántica.

Es importante resaltar que la herramienta se ha desarrollado para que pueda funcionar con diferentes tipos de textos y de idiomas para hacer la herramienta lo más independiente posible de los datos.

¹ <https://colab.research.google.com/>

² <https://es.reactjs.org/>

³ <https://huggingface.co/>

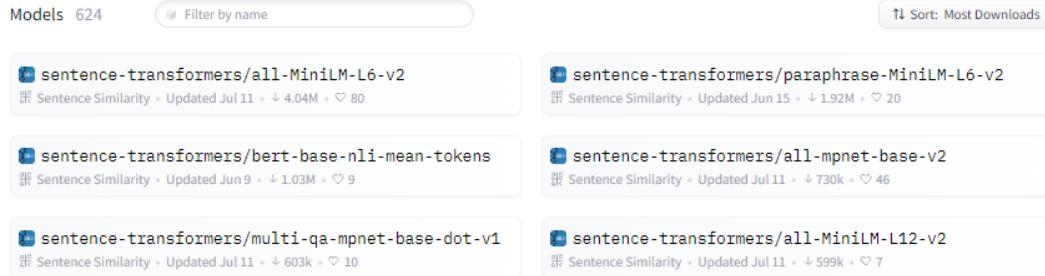


Figura 26. Captura de los modelos más descargados en Hugging Face.

Fuente: https://huggingface.co/models?pipeline_tag=sentence-similarity&sort=downloads

Juntando la eficacia que ofrecen estos modelos a la hora de producir *sentence-embeddings* con la potencia de cómputo que ofrece Google Colaboratory se puede llevar esta herramienta a cualquier usuario sin necesidad de tener una GPU propia para ejecutarlo. Esto se debe a que este servicio ofrece de manera gratuita un entorno de ejecución remoto en el que un usuario tiene a su disposición una GPU, que bastará para correr esta herramienta. En la Figura 27 podemos ver la información de nuestro sistema en cuenta gratuita y, en la Figura 28, podemos ver la información del sistema que nos proporcionan con la suscripción a Google Colaboratory Pro.

No es necesario la ejecución con aceleración por GPU, se puede ejecutar en CPU, pero el rendimiento se verá bastante afectado ya que la intención de esta herramienta es aprovechar al máximo la paralelización y esta segunda opción no favorece este objetivo.

```
▶ Comprobar la GPU y RAM disponible.

Esta celda comprueba si Google Colab le ha asignado una GPU o no y la cantidad de RAM que le ha concedido.

No es necesario disponer de una GPU, pero el uso de la misma hará que el código se ejecute mucho más rápido.

Mostrar código

Wed Aug 31 21:06:46 2022
+-----+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2 |
+-----+
| GPU  Name Persistence-M| Bus-Id Disp.A  Volatile Uncorr. ECC | | |
| Fan  Temp  Perf  Pwr/Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|                               |             |            | MIG M. |
+-----+
|   0  Tesla T4           Off  00000000:00:04.0 Off   0 |
| N/A   36C   P8    9W / 70W  0MiB / 15109MiB   0%      Default |
|                               |             |            | N/A |
+-----+
+-----+
| Processes:                   GPU Memory |
| GPU  GI  CI PID Type Process name        Usage  |
| ID  ID
+-----+
| No running processes found |
+-----+

Your runtime has 13.6 gigabytes of available RAM.

Not using a high-RAM runtime.
```

Figura 27. Información del sistema (GPU + RAM) asignado a cuenta gratuita.

▶ Comprobar la GPU y RAM disponible.

Esta celda comprueba si Google Colab le ha asignado una GPU o no y la cantidad de RAM que le ha concedido.

No es necesario disponer de una GPU, pero el uso de la misma hará que el código se ejecute mucho más rápido.

[Mostrar código](#)

```

Thu Sep 1 18:03:46 2022
+-----+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2 |
+-----+
| GPU  Name      Persistence-M| Bus-Id     Disp.A  | Volatile Uncorr. ECC | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| |          |          |              | MIG M.               |
+-----+
| 0  Tesla P100-PCIE... Off  | 00000000:00:04.0 Off |          0 | |
| N/A   36C   P0    27W / 250W |          0MiB / 16280MiB |  0%     Default |
|          |              |                  | N/A                |
+-----+
+-----+
| Processes:
| GPU  GI CI      PID  Type  Process name        GPU Memory |
| ID   ID          ID   ID    Usage
+-----+
| No running processes found
+-----+

```

Your runtime has 27.3 gigabytes of available RAM.

You are using a high-RAM runtime.

Figura 28. Información del sistema (GPU + RAM) asignado a cuenta Pro.

3.1.2. Arquitectura

La arquitectura de la herramienta está basada en diferentes elementos que pueden interactuar entre sí y cuyas funcionalidades básicas son las siguientes:

- **Inicializador:** prepara el directorio de trabajo para la ejecución y carga del conjunto de datos.
- **Bi-Encoder:** generación de los *sentence-embeddings* que representan a cada texto del conjunto de datos a explorar.
- **Cálculo de similitudes:** generación de la matriz de similitud entre los *sentence-embeddings*.
- **Cross-Encoder:** reordenamiento de los resultados obtenidos al calcular las similitudes para obtener un orden más acertado de los elementos.
- **Exploración de la matriz de similitud:** permite la exploración de las matrices de similitud generadas.
- **Exportar datos a una base de datos:** permite la exportación a una base de datos MySQL del conjunto de datos cargados junto a los resultados para soportar la exploración.

- **Exploración en tiempo real:** permite explorar el conjunto de datos de diferentes formas sin necesidad de calcular la matriz de similitud.
- **Exploración con otro conjunto de datos:** permite explorar el conjunto de datos con otro conjunto diferente.

En la Figura 30 podemos ver ilustrada la arquitectura y en la Figura 29 se puede ver la herramienta diseñada en Google Colaboratory, donde se muestran los distintos componentes en forma de celdas.

EXPLORACIÓN DE CONTENIDOS MULTIMEDIA BASADA EN LA SIMILITUD
SIMILARITY-BASED BROWSING OF MULTIMEDIA CONTENTS

- ▶ Primeros pasos: inicialización del entorno y carga del conjunto de datos a explorar

[] 4 3 celdas ocultas
- ▶ Cálculo o carga de los sentence embeddings y exploración en el espacio: Bi-Encoder.

En esta sección se calculan o cargan los sentence embeddings de los textos indicados del conjunto de datos.
 Para crear los embeddings existe una larga lista de modelos ya entrenados que se encuentran disponibles en [HuggingFace](#), se puede observar el rendimiento de los mismos [aqui](#).

[] 4 4 celdas ocultas
- ▶ Exploración en tiempo real.

En esta sección se puede explorar el conjunto de datos en tiempo real sin necesidad de tener calculada la matriz de similitud. Es necesario haber ejecutado la creación o el cargado de los sentence embeddings antes de ejecutar a la exploración.

[] 4 2 celdas ocultas
- ▶ Cálculo o carga de similitudes: Faiss.

En esta sección se calcula o carga la matriz de similitudes entre los sentence embeddings utilizando Faiss.

[] 4 2 celdas ocultas
- ▶ Re-ranking o carga de las similitudes: Cross-Encoder.

En esta sección se calcula o carga la matriz de similitudes entre los sentence embeddings utilizando el Cross-encoder.

[] 4 2 celdas ocultas
- ▶ Exploración de la matriz de similitud.

En esta sección se pueden explorar los resultados obtenidos anteriormente con el Bi-encoder o el Cross-encoder. Es necesario haber ejecutado al menos el cálculo de las similitudes con el Bi-encoder antes de lanzarte a la exploración.

[] 4 1 celda oculta
- ▶ Exportar dataframe a una base de datos

Se exporta el conjunto de datos cargados junto a las columnas que contienen los índices y similitudes de las más parecidas a cada elemento a una base de datos MYSQL.

[] 4 1 celda oculta
- ▶ Exploración con otro conjunto de datos.

En esta sección se puede explorar el conjunto de datos cargado con el que se le indica por parámetros. Es necesario haber ejecutado la inicialización del modelo y la creación o el cargado de embeddings del conjunto original antes de ejecutar a la exploración con el nuevo conjunto de datos.

[] 4 3 celdas ocultas

Figura 29. Componentes del sistema.

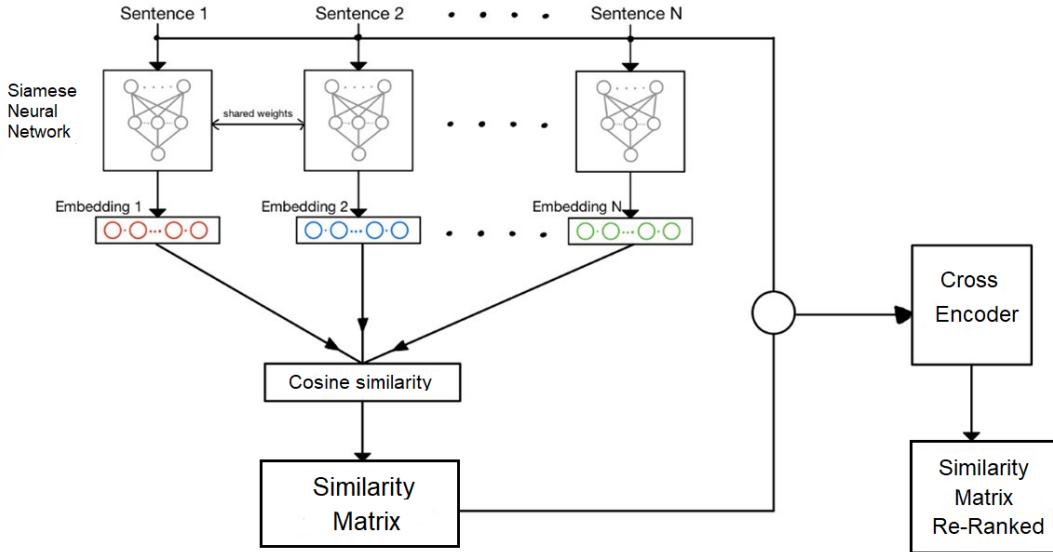


Figura 30. Arquitectura del cálculo de similitudes de la herramienta SBMC.

3.1.3. Inicialización del entorno y carga de datos

Para iniciar la exploración de un conjunto de datos primero se han de instalar las bibliotecas necesarias. Las más destacadas son:

- **Tensorflow¹**: biblioteca de código abierto para aprendizaje automático desarrollado por Google. Se utiliza para operar con tensores.
- **Numpy²**: biblioteca de código abierto especializada en el cálculo numérico con grandes volúmenes de datos.
- **Torch³**: biblioteca diseñada para realizar cálculos numéricos haciendo uso de la programación con tensores.
- **Pandas⁴**: biblioteca escrita como extensión de *Numpy* que permite la manipulación y análisis de datos. Ofrece estructuras de datos y operaciones que permiten manipular tablas numéricas y series temporales.
- **Sentence-transformers⁵**: biblioteca que permite usar técnicas del estado del arte del NLP. Está basado en *Torch* y *Transformers* y permite el acceso a los modelos entrenados de Hugging Face.

¹ <https://www.tensorflow.org/?hl=es-419>

² <https://numpy.org/>

³ <http://torch.ch/>

⁴ <https://pandas.pydata.org/>

⁵ <https://www.sbert.net/>

También es recomendable inicializar la GPU que nos ofrece Google Colaboratory ya que los cálculos serán mucho más rápidos.

Además, conectará con la cuenta de Google Drive¹ para almacenar y cargar los datos. En la Figura 31 se pueden ver las dos celdas que se encargan de las tareas mencionadas.

Importar e instalar dependencias.
Mostrar código

Comprobar la GPU y RAM disponible.

Esta celda comprueba si Google Colab le ha asignado una GPU o no y la cantidad de RAM que le ha concedido.

No es necesario disponer de una GPU, pero el uso de la misma hará que el código se ejecute mucho más rápido.

Figura 31. Instalación de dependencias e inicialización de la GPU.

En la Figura 32 se muestra la celda con la que se puede cargar el conjunto de datos sobre el que se desea realizar la exploración.

Iniciar el directorio de trabajo y cargar datos.

Para ejecutar el algoritmo conectaremos el cuaderno con Google Drive para almacenar la información. Esta celda conecta su cuenta drive con este notebook de Google Colab. Además inicializará el workspace necesario para ejecutar el código, creará 3 directorios: `similarity_matrix`, `saved_embeddings` y `saved_models` para almacenar las similitudes de los elementos, sus embeddings y los modelos que se deseen almacenar respectivamente.

Para ello se deben indicar los siguientes argumentos:

- **WS_PATH**: Ruta del directorio de trabajo. Es la carpeta en la que se crearán los directorios anteriormente mencionados. (**Ejemplo**: `/content/drive/general_explorer/`).
- **FILE_NAME**: Nombre del conjunto de datos que deseas explorar. Debe estar en la carpeta WS_PATH y debe de ser un fichero de formato csv. (**Ejemplo**: `booksummaries.csv`).
- **EMBD_COLUMN**: Nombre de la columna del conjunto de datos que contiene los textos sobre los que se realizará la exploración.
- **LOAD_FULL_CSV**: Si la casilla está marcada, se cargará el fichero indicado en FILE_NAME entero, en otro caso se deberá indicar el número de filas que desea cargar en el siguiente argumento.
- **NUMBER_OF_ROWS**: Número de filas que serán cargadas del archivo indicado en FILE_NAME.

```
WS_PATH: "/content/drive/Shareddrives/TFG_2021/final code/SMBC_EXPLORER/"  
FILE_NAME: "bundles_desc.csv.zip"  
EMBD_COLUMN: "description"  
LOAD_FULL_CSV:   
NUMBER_OF_ROWS: 2048
```

Figura 32. Inicialización del directorio de trabajo y carga de datos.

¹ https://www.google.com/intl/es_es/drive/

3.1.4. Cálculo de los *sentence-embeddings*: *Bi-Encoder*

Una vez inicializada la herramienta y cargado el conjunto de datos, el siguiente paso es seleccionar el *Bi-Encoder* que se va a utilizar para calcular los *sentence-embeddings* de los textos a comparar para, posteriormente, poder medir su semejanza con alguna medida como puede ser la similitud del coseno [23] o la distancia euclídea. Podemos ver en la Figura 33 la celda que se encarga de esta configuración.

De esta manera, dependiendo del tipo de datos que queramos explorar, podremos usar diferentes modelos que se pueden encontrar en Hugging Face y observar sus rendimientos en Hugging Face-Performances¹.

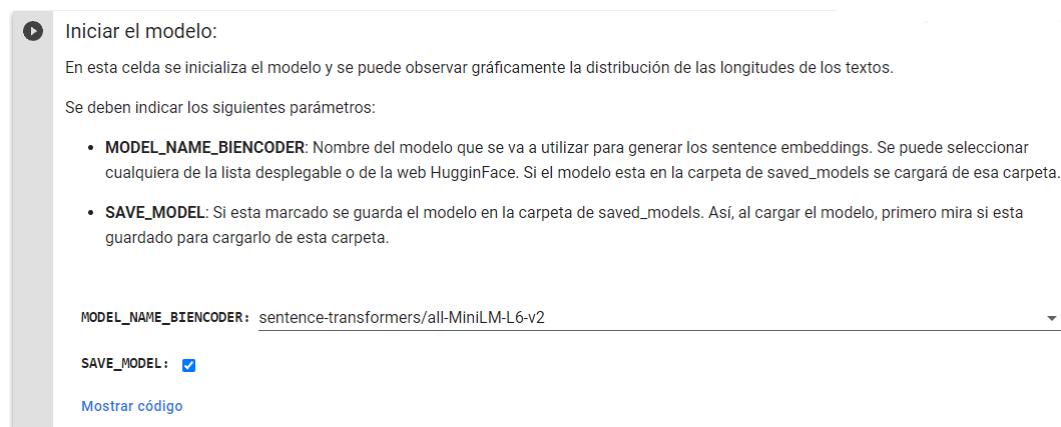


Figura 33. Inicialización del modelo.

Algunos de los modelos *Bi-Encoder* utilizados durante el proyecto son:

- **all-MiniLM-L6-v2²**: modelo basado en el *nreimers/MiniLM-L6-H384-uncased* de Microsoft [24] que ha pasado por un entrenamiento auto supervisado sobre un billón de pares de sentencias, sobre todo comentarios de Reddit³. Por defecto, trunca las oraciones de más de 256 palabras y genera *sentence-embeddings* en un espacio vectorial de 384 dimensiones. Se utiliza para capturar la información semántica para posteriormente clasificar o medir similitudes.
- **paraphrase-multilingual-MiniLM-L12-v2⁴**: es una versión del modelo *all-MiniLM-L12-v2⁵* que ha sido entrenado utilizando otro modelo multilenguaje permitiendo así una mayor capacidad de generalización.

¹ https://www.sbert.net/_static/html/models_en_sentence_embeddings.html

² <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

³ <https://www.reddit.com/>

⁴ <https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2>

⁵ <https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2>

Trunca los textos de más de 128 palabras y genera *sentence-embeddings* en un espacio de 384 dimensiones.

- ***all-mpnet-base-v2***¹: es una versión del modelo *mpnet-base*² de Microsoft que ha sido entrenado sobre un billón de pares de sentencias dando lugar al modelo en cuestión. Por defecto trunca las sentencias de más de 384 palabras y genera *sentence-embeddings* en un espacio de 768 dimensiones.
- ***bert-base-nli-mean-tokens***³: este modelo es uno de los primeros *Bi-Encoder* y está en desuso ya que ha sido superado por los anteriores. Por defecto trunca las oraciones de más de 512 elementos y representa estas sentencias en un espacio vectorial de 768 dimensiones.

De los anteriormente mencionados, cabe a destacar que el modelo *paraphrase-multilingual-MiniLM-L12-v2* nos permite explorar conjuntos de datos en más de 50 idiomas, lo que hace que la herramienta sea más independiente aún de los datos al dejar de depender del idioma de los textos lo cual es una ventaja ya que el inglés es el idioma principal sobre el que se han entrenado la mayoría de los modelos.

Para continuar, habrá que ejecutar la celda que calcula los *sentence-embeddings* con el modelo seleccionado o utilizar la siguiente que permite cargarlos si han sido previamente almacenados. Además, se dispone de otra celda que, aplicando técnicas de reducción de dimensionalidad, genera una visualización en 3D de los mismos en el espacio para poder hacerse una idea preliminar de cómo están estructurados. Se puede ver ilustrado en la Figura 34.

¹ <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

² <https://huggingface.co/microsoft/mpnet-base>

³ <https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens>

▶ Calcular los sentence embeddings.

En la celda actual se deben indicar los siguientes parámetros:

- **MAX_SEQ_LEN** : Maxima longitud de los textos a procesar. Textos con un número mayor de palabras serán truncados. Si es -1 se usará el valor por defecto del modelo seleccionado. Puedes ayudarte de la gráfica anteriormente generada para ajustar este valor. ([Ejemplo: 512](#)).
- **NORMALIZE_EMBEDDINGS** : Si está marcada los embeddings serán normalizados. Recomendable si se quiere obtener similitudes entre 0 y 1.
- **SAVE_EMBEDDINGS** : Si está marcada los embeddings serán guardados con el nombre indicado en el parametro OUTPUT_NAME_EMBEDDINGS.
- **OUTPUT_NAME_EMBEDDINGS** : Nombre con el que serán guardados los embeddings. ([Ejemplo: embeddings-peliculas](#))

MAX_SEQ_LEN:

NORMALIZE_EMBEDDINGS:

SAVE_EMBEDDINGS:

OUTPUT_NAME_EMBEDDINGS:

[Mostrar código](#)

- ▶ Cargar embeddings.
- EMBEDDINGS_FILE_NAME:**
- [Mostrar código](#)
- ▶ Explorar embeddings en el espacio.

Figura 34. Cálculo o carga de los sentence-embeddings.

3.1.5. Cálculo de las similitudes: *Faiss*

Una vez se tienen los *sentence-embeddings*, podemos calcular la matriz de similitud entre los elementos del conjunto.

En una primera implementación se desarrolló un algoritmo iterativo para calcular una matriz numérica de tantas filas como componentes de la colección y tantas columnas como número de similares que queremos guardar. Calcular las similitudes entre todos los componentes a la vez es inviable por el espacio en memoria de GPU que requiere. Por ello, en cada iteración se calculan las similitudes de un subconjunto de ellos con todas las demás y se guardan el número de similares deseado. Guardar la similitud con todos los componentes sería un gran gasto de memoria, ya que no es frecuente que un componente tenga muchos otros similares y realmente los resultados relevantes se mantienen en un conjunto reducido de ellos.

Para ello, en un principio se decidió usar la función *topk*¹ de la biblioteca *Torch*, que permite obtener el número deseado de elementos más similares a un *sentence-embedding* proporcionado. Además, se encuentra optimizada para el cálculo con tensores y además permite la simplificación del código manteniendo el mismo enfoque.

En la implementación final se decidió usar *Faiss*, una biblioteca de Facebook AI orientada a la búsqueda de vectores similares que redujo el tiempo de ejecución de varios minutos a apenas segundos obteniendo prácticamente los mismos resultados. Este algoritmo se basa en la construcción de una estructura de datos en RAM denominada *index*, que contendrá nuestros *sentence-embeddings*. De esta manera, cuando queremos obtener los elementos más similares a un *sentence-embedding* concreto, se calcula el producto escalar de este con todos los que se encuentran en el *index*. En caso de que los *sentence-embeddings* se encuentren normalizados, *Faiss* devuelve la similitud del coseno. Esto es recomendable porque además de tener ordenados los elementos de mayor a menor similitud, estos valores son fácilmente interpretables al estar en un rango entre -1 y 1.

Faiss proporciona diferentes algoritmos de indexación dependiendo de si queremos exactitud o rapidez, además de poder elegir si ejecutarlo en GPU o CPU. Algunos se basan en la compresión de vectores para aumentar la velocidad de ejecución de este, mientras que otros mantienen el vector original entero a costa de memoria y velocidad, pero mejorando la búsqueda.

En este caso es de interés conseguir los mejores resultados y mantener el vector original, ya que el objetivo es tener ya calculadas las similitudes para que cuando el usuario navegue por las componentes de la colección este pueda hacerlo de la manera más rápida posible aunque, realmente, se podrían calcular las similitudes en tiempo real ya que teniendo los *sentence-embeddings* calculados la obtención de los similares a un elemento dado es casi instantánea.

¹ <https://pytorch.org/docs/stable/generated/torch.topk.html>

Así, para la ejecución de este componente con *Faiss* basta con ejecutar la celda de la Figura 35 rellenando los parámetros indicados que permiten personalizar el guardado de la matriz

También existe una opción de cargar la matriz por si se tenía calculada y no se quiere volver a calcular.

▼ Cálculo o carga de similitudes: Faiss.

En esta sección se calcula o carga la matriz de similitudes entre los sentence embeddings utilizando *Faiss*.

⌚ Cálculo de similitudes - Bi-encoder

Esta celda permite calcular la matriz de similitudes. Para ello se deben de indicar los siguientes parámetros:

- **NUM_SAVE**: Número de elementos similares que se quieren guardar para cada uno de los elementos del conjunto de datos.
- **SAVE_SIMILARITY_MATRIX_PICKLE**: Si la casilla está marcada la matriz de similitudes será guardada en formato .pkl con el nombre indicado en OUTPUT_NAME. Si se quieren cargar posteriormente en este cuaderno se deben de guardar con este formato.
- **SAVE_SIMILARITY_MATRIX_CSV**: Si la casilla está marcada la matriz de similitudes será guardada en formato .csv con el nombre indicado en OUTPUT_NAME. Usar cuando se quiera exportar los datos a otra plataforma en la que utilizar el archivo en formato csv.
- **OUTPUT_NAME**: Nombre del fichero en el que se guardará la matriz de similitudes generada.

NUM_SAVE: 50

SAVE_SIMILARITY_MATRIX_PICKLE:

SAVE_SIMILARITY_MATRIX_CSV:

OUTPUT_NAME: "similarity_matrix-cellphones"

[Mostrar código](#)

⌚ Cargar matriz de similitudes - Bi-encoder.

Si se tiene la matriz de similitud ya calculada, se puede cargar el fichero con la misma y evitar tener que calcularla de nuevo. Indicar el nombre del fichero (en formato .pkl):

MATRIX_FILE_NAME: "similarity_matrix_cellphones"

Figura 35. Cálculo de similitudes: *Faiss*, carga de similitudes, y exploración en el espacio.

3.1.6. Reclasificación de soluciones: *Cross-Encoder*

Como ya se ha mencionado, el *Cross-Encoder* es considerablemente más lento que el *Bi-Encoder* ya que recibe los dos textos a la vez y nos devuelve directamente su similitud sin producir *sentence-embeddings*. De este modo, habría que pasarle todos los posibles pares de texto y, aunque es más preciso para calcular la similitud semántica que el *Bi-Encoder*, su coste computacional es mucho más elevado. Por ello, lo que se hace es que inicialmente se realiza una comparación entre las descripciones textuales de todos los componentes usando el *Bi-Encoder*, quedándonos con un número reducido de similares a cada uno como se ha descrito en el punto anterior para, posteriormente, ser reclasificadas utilizando el *Cross-Encoder* obteniendo así una nueva ordenación y puntuación para los resultados obtenidos anteriormente.

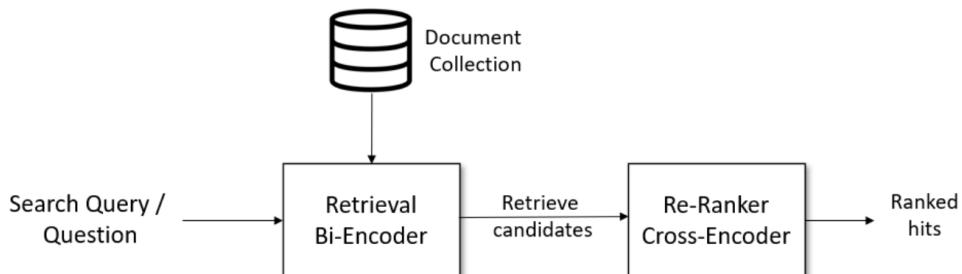


Figura 36. Esquema de reclasificación de soluciones.

Fuente: https://www.sbert.net/examples/applications/retrieve_rerank/README.html

Al igual que los *Bi-Encoder*, tenemos muchos *Cross-Encoder* ya entrenados que se pueden encontrar en Hugging Face. Algunos de los modelos *Cross-Encoder* utilizados durante el proyecto son:

- ***stsbert-roberta-large*¹**: este modelo fue entrenado sobre el STS *Benchmark dataset* y devuelve valores del 0 al 1 indicando la similitud entre dos sentencias.
- ***stsbert-roberta-base*²**: es la versión reducida del modelo anterior, por lo que reduce los tiempos de ejecución.
- ***stsbert-TinyBERT-L-4*³**: también fue entrenado sobre el STS *Benchmark dataset*. Sin embargo, éste es bastante más pequeño y los tiempos de ejecución se reducen drásticamente sin perder mucha precisión.

¹ <https://huggingface.co/sentence-transformers/stsb-roberta-large>

² <https://huggingface.co/sentence-transformers/stsb-roberta-base>

³ <https://huggingface.co/sentence-transformers/stsb-TinyBERT-L-4>

- ***ms-marco-MiniLM-L-12-v2***¹: este modelo se puede utilizar como buscador de información. Dada una consulta, ésta es codificada y se generan las posibles respuestas a la misma.

En la Figura 37 se puede ver una captura de este componente y los parámetros que se han de indicar para la ejecución.

▼ Re-ranking o carga de las similitudes: Cross-Encoder.

En esta sección se calcula o carga la matriz de similitudes entre los sentence embeddings utilizando el Cross-encoder.

➊ Re-ranking de resultados : Cross-encoder.

En esta sección se obtendrá un nuevo orden (re-ranking) para los resultados obtenidos con el Bi-encoder. Este tipo de modelo es mucho más lento pero más preciso, por lo que se recomienda que el número de elementos similares que se desean re-ordenar de cada elemento del conjunto de datos no sea muy grande. (**No se recomienda ejecutar sin disponer de una GPU**)

En la esta celda se inicializa el modelo Cross-encoder a utilizar.

Se deben llenar los siguientes parámetros:

- **MODEL_NAME_CROSSENCODER**: Nombre del modelo Cross-encoder que se va a utilizar.
- **NUM_SAVE**: Número de elementos similares a cada elemento obtenidos por el Bi-encoder que se desean re-rankear.
- **SAVE_SIMILARITY_MATRIX_PICKLE**: Si la casilla está marcada la matriz de similitudes será guardada en formato .pkl con el nombre indicado en OUTPUT_NAME. Si se quieren cargar posteriormente en este cuaderno se deben de guardar con este formato.
- **SAVE_SIMILARITY_MATRIX_CSV**: Si la casilla está marcada la matriz de similitudes será guardada en formato .csv con el nombre indicado en OUTPUT_NAME. Usar cuando se quiera exportar los datos a otra plataforma en la que utilizar el archivo en formato csv.
- **OUTPUT_NAME**: Nombre del fichero en el que se guardará la matriz de similitudes generada.

```
MODEL_NAME_BIENCODER: "cross-encoder/ms-marco-MiniLM-L-12-v2"
NUM_SAVE: 2
SAVE_SIMILARITY_MATRIX_PICKLE: 
SAVE_SIMILARITY_MATRIX_CSV: 
OUTPUT_NAME: "cross-encoder-cellphones-ms-marco-MiniLM-L-12-v2"
```

[Mostrar código](#)

➋ Cargar matriz de similitudes : Cross-encoder.

Si se tiene la matriz de similitud ya calculada, se puede cargar el fichero con la misma y evitar tener que calcularla de nuevo. Indicar el nombre del fichero (en formato .pkl):

```
MATRIX_FILE_NAME: "cross-encoder-cellphones-ms-marco-MiniLM-L-12-v2"
```

Figura 37. Reclasificación o carga de similitudes: Cross-Encoder.

¹ <https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2>

3.1.7. Exploración de la matriz de similitud

Una vez obtenida la matriz de similitud se puede ejecutar este componente para examinar los elementos similares recuperados a un elemento dado del conjunto. Se podrá elegir tanto realizar la exploración sobre la matriz obtenida por el *Bi-Encoder* como la obtenida por el *Cross-Encoder*.

En la Figura 38 se puede ver la celda que realiza lo mencionado.

Exploración de la matriz de similitud.

En esta sección se pueden explorar los resultados obtenidos anteriormente con el Bi-encoder o el Cross-encoder. Es necesario haber ejecutado al menos el cálculo de las similitudes con el Bi-encoder antes de lanzarte a la exploración.

- ➊ Obtener similares a un elemento dado.
 - BI_CO: Indica el encoder sobre el que quieras explorar los resultados.
 - INDEX_IN_CSV: Indica el índice del elemento del que deseas explorar las similares.
- BI_CO: Bi-encoder
- INDEX_IN_CSV: 0

Figura 38. Exploración de la matriz de similitud.

3.1.8. Exportación de datos

Este componente permite exportar a una base de datos el conjunto de datos explorado junto a la matriz de similitud obtenida en la exploración.

Como se puede ver en la Figura 39, se pueden indicar en la celda los parámetros de conexión a la base de datos y el nombre de la tabla en la que se quiere guardar para realizar esta exportación de manera automática.

Exportar dataframe a una base de datos

Se exporta el conjunto de datos cargados junto a las columnas que contienen los índices y similitudes de las más parecidas a cada elemento a una base de datos MySQL.

- ➊ • MYSQL_HOST: Dirección del host de la DB. (Ejemplo: 82.54.219.71:3306 o localhost)
 - MYSQL_USER: Usuario de la DB (Ejemplo: root).
 - MYSQL_PASSWORD: Contraseña del usuario de la DB.
 - MYSQL_DB: Nombre del schema de la DB.
 - MYSQL_TABLE_NAME: Nombre de la tabla a la que se desea exportar los datos.
- MYSQL_HOST: "localhost"
- MYSQL_USER: "root"
- MYSQL_PASSWORD: "localhost"
- MYSQL_DB: "bd_tfg"
- MYSQL_TABLE_NAME: "tfg"

Figura 39. Exportación de datos a una base de datos.

3.1.9. Exploración en tiempo real

Este componente ofrece diferentes formas de navegación por la colección de datos permitiendo encontrar elementos similares sin necesidad de tener calculada la matriz de similitud: basta con tener los *sentence-embeddings* calculados.

Para navegar sobre la colección en tiempo real disponemos de dos opciones:

- Exploración de los elementos del conjunto dado un índice de este y eligiendo el número de elementos similares a recuperar como se puede ver en la Figura 40.
- Exploración de los elementos del conjunto dado un texto proporcionado por parámetro en la celda y eligiendo el número de elementos similares a recuperar como se puede ver en la Figura 40.

▼ Exploración en tiempo real.

En esta sección se puede explorar el conjunto de datos en tiempo real sin necesidad de tener calculada la matriz de similitud. Es necesario haber ejecutado la creación o el cargado de los sentence embeddings antes de ejecutar la exploración.

► Explorar uno con todos.

- **INDEX_IN_CSV** : Indica el index del elemento del que se desea obtener sus similares.
- **NUM_SAVE** : Número de elementos similares que se desea obtener.

INDEX_IN_CSV: 45453

NUM_SAVE: 50

[Mostrar código](#)

► Explorar todos con tu propio texto.

- **TEXT** : Texto con el que se desea comparar los elementos del conjunto de datos.
- **NUM_SAVE** : Número de elementos similares que se desean obtener.

TEXT: "mild flavored coffee"

NUM_SAVE: 50

Figura 40. Exploración en tiempo real.

3.1.10. Exploración con otro conjunto de datos

Además de poder explorar los elementos de un conjunto entré si, también es de interés disponer de este componente que permite realizar una comparación entre dos conjuntos de datos.

En la Figura 41 se puede ver la celda que se encarga de esta ejecución y los parámetros que se han de indicar.

Exploración con otro conjunto de datos.

En esta sección se puede explorar el conjunto de datos cargado con el que se le indica por parámetros. Es necesario haber ejecutado la inicialización del modelo y la creación o el cargado de embeddings del conjunto original antes de ejecutar a la exploración con el nuevo conjunto de datos.

Se deben indicar los siguientes parámetros:

- **DF_NAME**: Nombre del conjunto de datos con el que se desea comparar el actual. Debe estar en la carpeta WS_PATH y debe de ser un fichero de formato csv. (**Ejemplo**: `booksummaries.csv`).
- **EMBD_COLUMN_COMPARE**: Nombre de la columna del conjunto de datos que contiene los textos que se compararan con el conjunto actual.
- **NUM_SAVE** : Número de elementos similares que se desea obtener.
- **SAVE_SIMILARITY_MATRIX_PICKLE**: Si la casilla está marcada la matriz de similitudes será guardada en formato .pkl con el nombre indicado en **OUTPUT_NAME**. Si se quieren cargar posteriormente en este cuaderno se deben de guardar con este formato.
- **SAVE_SIMILARITY_MATRIX_CSV**: Si la casilla está marcada la matriz de similitudes será guardada en formato .csv con el nombre indicado en **OUTPUT_NAME**. Usar cuando se quiera exportar los datos a otra plataforma en la que utilizar el archivo en formato csv.
- **OUTPUT_NAME** : Nombre con el que serán guardada la matriz de similitudes. (**Ejemplo**: `similitudes_elementos`)

DF_NAME:	<code>"bad_overviews_dataset.csv"</code>
EMBD_COLUMN_COMPARE:	<code>"Bad_overviews"</code>
NUM_SAVE:	<code>2048</code>
SAVE_SIMILARITY_MATRIX_PICKLE:	<input checked="" type="checkbox"/>
SAVE_SIMILARITY_MATRIX_CSV:	<input checked="" type="checkbox"/>
OUTPUT_NAME:	<code>"similarity with bad overviews_31_08"</code>

Figura 41. Exploración con otro conjunto de datos.

3.2. Explorador de películas basado en la similitud

3.2.1. Introducción

Para poner a prueba el explorador de contenido desarrollado se han obtenido datos sobre películas de la página web TMDB y los hemos usado en la herramienta para generar la matriz de similitud y exportarla a una base de datos. Así, se desarrolla una aplicación web que hace uso de esta base de datos para permitir al usuario navegar por las películas basándose en la similitud semántica de las sinopsis.

3.2.2. Arquitectura

El *back-end* está compuesto por el algoritmo generador de similitudes, la base de datos y el servidor que nos permite hacer consultas a la misma. El algoritmo se realizó en *Python* usando *Flask*¹ debido a su gran compatibilidad con todos los sistemas operativos y la cantidad de bibliotecas y *frameworks* útiles que nos proporciona en el campo de nuestro proyecto. Por otro lado, el servidor tiene que ser capaz de ejecutar el algoritmo en tiempo real y por ello se decidió realizar también en *Python*, de modo que la integración con el algoritmo fuera muy sencilla. Por último, la base de datos se ha realizado en *SQL* utilizando el gestor de bases de datos *MySQL Workbench*².

Para lograr una web en la que se pueda navegar por las distintas películas es necesario conectar distintos componentes encargados de realizar diferentes funciones, desde la obtención de los datos, pasando por el tratamiento de estos hasta su visualización por pantalla.

El *front-end* por su parte, está formado por una web con la que el usuario puede interactuar para navegar por las diferentes películas y encontrar relaciones entre estas. Se desarrolló en *React*³ (basado en *JavaScript*) debido a su rapidez y gran capacidad de escalabilidad y mantenimiento. También se le dio vistosidad y dinamismo a la interfaz utilizando *CSS*.

¹ <https://flask.palletsprojects.com/en/2.2.x/installation/#python-version>

² <https://www.mysql.com/products/workbench/>

³ <https://es.reactjs.org/>

3.2.3. Extracción de datos

Este es el componente inicial de nuestro proyecto. Al principio, se intentó obtener los datos de las películas a través de la web de IMDB¹, la mayor base de datos de películas pública en el mundo. Sin embargo, esta web no disponía de una API pública, por lo que se decidió utilizar los datos de la web de TMDB que sí disponía de ella.

Tras obtener la clave API y el marco de datos con los identificadores de las películas disponibles en TMDB, se realizaron las peticiones correspondientes de cada uno de los identificadores obtenidos de TMDB, los *Daily File Exports*² que proporciona la lista de identificadores de películas disponibles.

Esta solución era muy costosa en tiempo, por lo que se utilizan técnicas de multiprocesamiento para ejecutar esta tarea en paralelo y reducir de forma considerable el tiempo de ejecución.

Se obtuvieron muchos datos de cada película, muchos de ellos irrelevantes, por lo que finalmente nos quedamos con la siguiente información: el identificador, los géneros, los actores y actrices, el título original, la sinopsis, el enlace del poster, la fecha de estreno, y la duración total de la película.

Gracias a este componente, finalmente se obtuvo la información de aproximadamente un millón de películas con las se trabajó en todo momento. Cabe destacar que con cada ejecución se puede obtener una versión actualizada del conjunto de películas.

Las celdas de código que se encargan de la obtención de datos desde TMDB se encuentran ilustradas en la Figura 42.

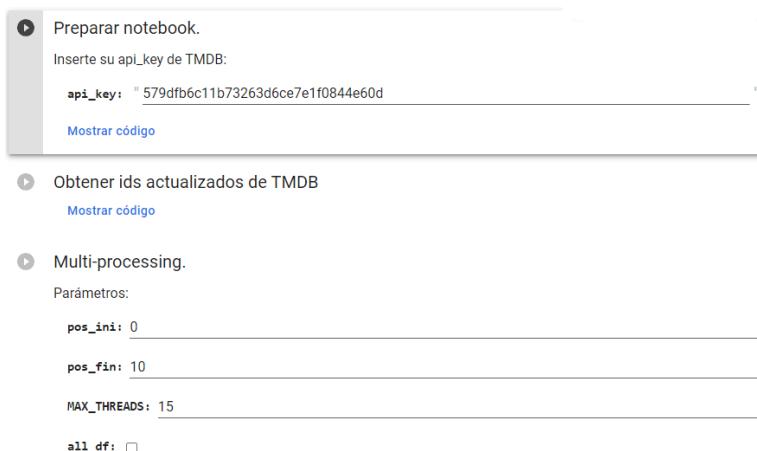


Figura 42. Celdas encargadas de la obtención de datos.

¹ <https://www.imdb.com/>

² <https://developers.themoviedb.org/3/getting-started/daily-file-exports>

3.2.4. Preprocesamiento de datos

Una vez obtenidos los datos, se hizo una limpieza de estos. Esto se debe a que la información de las películas es proporcionada por los propios usuarios y muchas de las películas obtenidas, o bien no tenían sinopsis con la que realizar la comparación, o bien no era relevante para la exploración. Para obtener un conjunto de datos con información útil para el algoritmo se siguieron una serie de pasos:

1. Se eliminaron aquellas películas que en los campos *title* o *overview* solo estuvieran compuestos por espacios, tabuladores, saltos de línea, etc.
2. Gracias a un estudio previo se eliminó el argumento no válido más repetido, “*No overview found*”.
3. Se estudió la longitud de los argumentos de las películas y se suprimieron aquellas que no contenían un mínimo de palabras necesarias.
4. Se detectó el idioma de las sinopsis utilizando la biblioteca *langdetect*¹ y se borraron aquellas que no estuvieran en inglés, ya que la mayoría de los argumentos están en este idioma y mejorará la eficacia del modelo a la hora de compararlos.

Las celdas del *notebook* que hay que ejecutar para realizar los pasos anteriormente explicados se pueden observar en la Figura 43.

▼ Eliminar argumentos no útiles.

► Remplazamos los elementos que solo contengan espacios vacíos por NaN y borramos todos los NaN.
[Mostrar código](#)

► Borramos los "No overview found.".
[Mostrar código](#)

► Estudiar longitud de los overview.
[Mostrar código](#)

► Borrar los elementos que tengan un argumento con un numero de palabras muy reducido.
[Mostrar código](#)

► Borrar sentencias que no están en inglés. (Aprox: 1h)
[Mostrar código](#)

Figura 43. Borrado de argumentos inválidos.

¹ <https://pypi.org/project/langdetect/>

5. Se usó el SBMC desarrollado para medir la similitud con las sinopsis no relevantes que se han predefinido para así encontrar otras inválidas como por ejemplo “*no overview*”, “*no plot*” o “*no description available*” y poder eliminarlas también si superan cierto criterio de similitud. Para ello se utiliza el componente de la herramienta que permite comparar un conjunto de datos con otro. De esta forma, generando uno que contenga las sinopsis no validas podemos encontrar varias similares a las del conjunto de películas de forma sencilla y rápida. Podemos ver un ejemplo de las sinopsis no válidas encontradas en la ejecución de este paso en la Figura 44.

Eliminar argumentos de películas similares a los no útiles ya conocidos - SMBC.

- ▶ Generar dataframe de overviews no útiles.

[Mostrar código](#)

- ▶ Tras usar la comparación de dataframes en el SMBC cargamos la matriz de similitud obtenida.


```
MATRIX_FILE_NAME: "similarity with bad overviews_31_08.pkl"
```


[Mostrar código](#)

- ▶ Borramos los overviews con alta similitud a los bad_overviews.

[Mostrar código](#)

index		index	
467994	Plot synopsis coming at a later date	378979	no overview at the moment
301666	No Plot as of Yet.	84334	No overview is available.
475175	Plot is currently unknown	577336	there is no any overview
486758	No plot found online.	485183	No Overview details at this time
254590	Plot is under wraps.	525328	Overview is unknown at this time
418741	the plot is unknown	516218	No English overview available now.
191972	The plot is not yet known.	517389	No English overview available at this time.
486743	No plot found anywhere online.	194529	English Overview is not available
493796	Plot unknown at this time.	221768	no official overview for now.
575222	Plot Unknown At This Time.	319994	(no summary available right now)
481566	Plot is unknown at this time.	205896	No movie overview available
374031	Movie Plot Not Provided.	238688	Need to add description
507213	The plot isn't known at this time.	47749	No movie overview available.
561872	No plot can be found for this film.	319836	(no summary for the moment)
283857	The plot is unknown.	120749	No movie overview is available.
223841	No plot details have been announced.	233518	No description could be found.
559159	Short film, 11 minutes long, no plot currently available	583231	Description Unknown at time
479634	Still missing a plot...	250120	sorry we don't have description
303018	Plot not Known at this Certain Time. Will be a synopsis soon.	412038	No synopsis and plot is known
583422	Plot kept under wraps.	486704	No plot synopsis found anywhere online.
244195	The plot is currently unknown.	294894	No synopsis. Released 1911, date Unknown.
317236	Status and plot is at this moment unknown.	571520	Cannot find any synopsis of this film.
517953	Short film. Plot unknown.	197773	No movie summary available

Figura 44. Sinopsis no validas encontradas durante el preprocesamiento del conjunto.

3.2.5. Obtención de similitudes usando SBMC

Una vez tenemos nuestro conjunto de datos procesado, obtenemos un *dataset* final con la información de 584.500 películas diferentes. Este servirá de entrada al SBMC para obtener nuestra matriz de similitud.

Para ello, el primer paso es ejecutar la celda de la Figura 31 que se encarga de instalar las dependencias e iniciar la GPU. Una vez realizado, se procede a cargar el conjunto de películas e iniciar el directorio de trabajo ejecutando la celda de la Figura 32.

A continuación, se elige el modelo *all-MiniLM-L6-v2* para calcular los *sentence-embeddings* de las sinopsis mediante las ejecuciones de las celdas de la Figura 33 y Figura 34. Se ha elegido este modelo en concreto porque está entrenado en inglés, como la gran mayoría de nuestras películas y su precisión junto con el tiempo de ejecución es muy buena comparada con otros modelos. Podemos observar una comparación de los tiempos de ejecución de distintos modelos y distintos procesadores en la Figura 45.

Modelo	Tesla P100	Tesla T4	CPU (estimado)
all-MiniLM-L6-v2	5' 23"	8' 32"	10h 29' 18"
all-mpnet-base-v2	24' 39"	48' 43"	63h 42' 20"
paraphrase-MiniLM-L6-v2	4' 42"	7' 35"	6h 5' 23"
multi-qa-mpnet-base-dot-v1	24' 37"	49' 33"	60h 15' 11"
paraphrase-multilingual-MiniLM-L12-v2	9' 32"	14' 37"	8h 56' 43"

Figura 45. Comparación de modelos con distinto hardware.

Tras aproximadamente 10 minutos de espera, una vez obtenidos los *sentence-embeddings* se puede empezar a calcular la similitud entre estos usando *Faiss* mediante la ejecución de la celda ilustrada en la Figura 35, la cual no tarda más de 5 minutos.

El siguiente paso es realizar una reclasificación de las similares a cada elemento para obtener un nuevo orden más acertado. Para ello se ejecuta la celda de la Figura 37 configurada para utilizar el *Cross-Encoder stsbert-TinyBERT-L-4*.

Para terminar, se exporta a la base de datos mediante el componente de la Figura 39 que facilita esta tarea. Una vez ha terminado, los datos la colección están preparados para ser explorados utilizando la interfaz web.

Todo este proceso se realiza en aproximadamente 40 minutos, de los cuales 20 son para el cálculo de los *sentence-embeddings* y las similitudes y otros 20 para la exportación de los resultados a la base de datos.

El SBMC también se utiliza para realizar el cálculo de los *sentence-embeddings*, tanto de los títulos como de los actores que, posteriormente, son utilizados en el servidor para realizar búsquedas en tiempo real ejecutando el algoritmo.

3.2.6. Entrenamiento de modelos

Hasta ahora únicamente se han utilizado modelos preentrenados y combinación de ellos, pero se han desarrollado tambien otros enfoques continuando el entrenamiento de estos sobre nuestro conjunto de datos de peliculas para ver si tenía efecto sobre los resultados. A continuación se detallan algunos de los entrenamientos que hemos realizado.

3.2.6.1. Primera versión

En un principio, para mejorar las predicciones del *Bi-Encoder* se usa un *Cross-Encoder* ya entrenado para etiquetar una porcion de nuestros datos con similitudes calculadas por este y poder usarlo para entrenar al *Bi-Encoder* como se ilustra en la Figura 46. Este proceso esta descrito en la publicacion *Domain Transfer with Augmented SBERT* [25].

Ya que en este caso de las películas no existen conjuntos de datos etiquetados con su similitud con otras, este enfoque permitiría generar un conjunto de datos sintético con el que intentar traspasar el conocimiento de un *Cross-Encoder* a un *Bi-Encoder*. Después de probar a entrenar al modelo con diferentes parámetros, se descubrió que los resultados obtenidos no mejoraban.

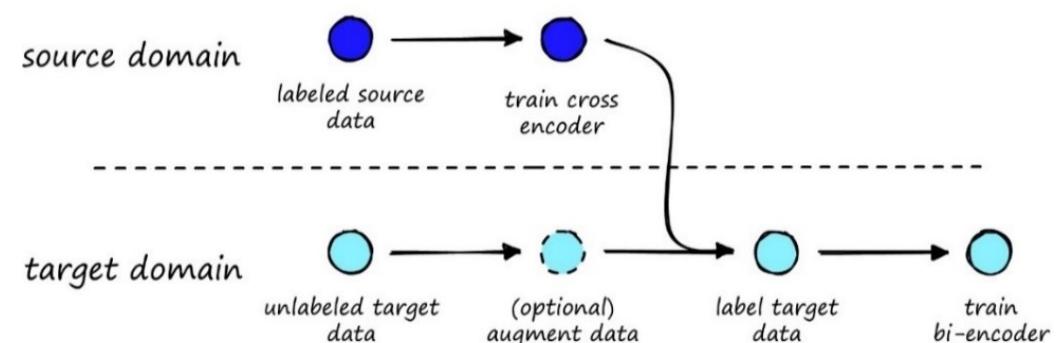


Figura 46. Domain Transfer.

Fuente: <https://www.pinecone.io/learn/domain-transfer/>

3.2.6.2. Segunda versión

En un segundo intento se utiliza un método de entrenamiento llamado *Domain Adaptation*¹, que tiene como objetivo adaptar los *word-embeddings* que produce un modelo a un dominio específico (en nuestro caso synopsis de películas) utilizando datos sin etiquetar de este junto a otro conjunto de datos etiquetados que se pueden obtener en Hugging Face. En la Figura 47 se pueden ver algunos de estos conjuntos, junto con su descripción, numero de ejemplos de diferentes dominios y fuentes.

Dataset	Description	(#Lines)	Performance
gooaq_pairs.jsonl.gz	(Question, Answer)-Pairs from Google auto suggest	3,012,496	59.06
yahoo_answers_title_answer.jsonl.gz	(Title, Answer) pairs from Yahoo Answers	1,198,260	58.65
msmarco-triplets.jsonl.gz	(Question, Answer, Negative)-Triplets from MS MARCO Passages dataset	499,184	58.76
stackexchange_duplicate_questions_title_title.jsonl.gz	(Title, Title) pairs of duplicate questions from StackExchange	304,525	58.47
el15_question_answer.jsonl.gz	(Question, Answer)-Pairs from EL15 dataset	325,475	58.24
yahoo_answers_title_question.jsonl.gz	(Title, Question_Body)	659,896	58.05

Figura 47. Datasets disponibles en Hugging Face.

Fuente: <https://huggingface.co/datasets/sentence-transformers/embedding-training-data>

Para realizar este tipo de entrenamiento hemos utilizado un método denominado GenQ [26]. Cómo se puede ver en la Figura 48, este método utiliza un modelo T5 [27] que esta entrenado para generar preguntas sobre un texto dado y así conseguir un conjunto de datos etiquetados de consultas y resultados. De esta forma, se generarán 5 consultas para cada sinopsis, para tener la información de esta representada como preguntas y utilizar esta tupla de consultas y resultado para entrenar al *Bi-Encoder* para capturar la información semántica entre esas tuplas.

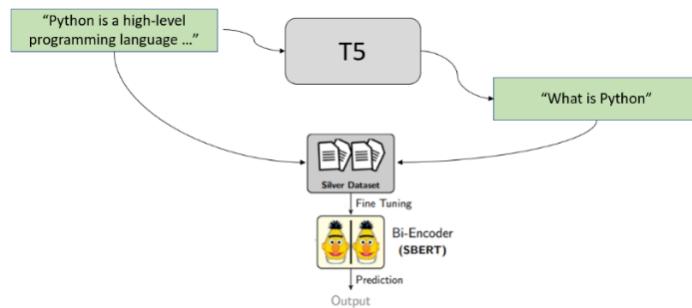


Figura 48. Pasos del método GenQ.

Fuente: https://www.sbert.net/examples/unsupervised_learning/README.html#genq

¹ https://www.sbert.net/examples/domain_adaptation/README.html

Para realizar el entrenamiento del modelo, se debe de definir una función de error que determine cómo de bien funciona el modelo. En este caso se usará la función *MultipleNegativeRankingLoss*¹ [28].

Esta función espera un conjunto de pares de texto de la forma $[(a_1, b_1)], \dots, (a_n, b_n)$ donde se asumen que (a_i, b_i) son similares, mientras que (a_i, b_j) con $i \neq j$ no lo son; para a continuación minimizar la distancia entre los a_i y b_i , y maximizarla entre los a_i y b_i con $i \neq j$, cómo se ilustra en la Figura 49.

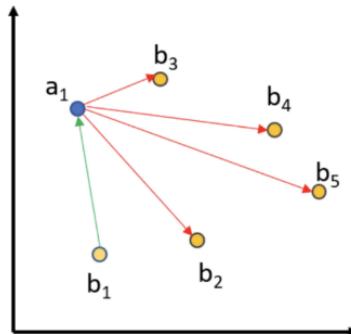


Figura 49. Ilustración del método *MultipleNegativeRankingLoss*.

Fuente: https://www.sbert.net/examples/training/ms_marco/README.html

3.2.6.3. Tercera versión

Durante el desarrollo del proyecto aparece GPL [29] (del inglés Generative Pseudo Labeling), una nueva técnica que mejoraba a la GenQ, por lo que se decide implementar este tipo de entrenamiento. Este funciona en 3 fases como se puede ver en ilustrado en la Figura 50:

- **Query Generation:** para un texto de nuestro dominio, usamos un modelo T5 igual que el que se ha mencionado antes: se generan preguntas que se responden en el texto dado. Se pueden encontrar diferentes modelos en Hugging Face².
- **Negative Mining:** una vez generadas la pregunta, se buscan textos relacionados con esta usando uno de los modelos ya entrenados para generar los *sentence-embeddings* de las preguntas y encontrar textos que estén relacionados, pero que un usuario no consideraría relevante porque no responde realmente a la pregunta.
- **Pseudo Labeling:** para no seleccionar un texto como irrelevante que realmente sea relevante, se usa un *Cross-Encoder* para etiquetar con similitudes todos los pares (*query, passage*). En métodos anteriores se utilizaba solo 2 valores para etiquetar como positiva (1) o negativa (0), y

¹ https://www.sbert.net/docs/package_reference/losses.html#multiplenegativesrankingloss

² <https://huggingface.co/doc2query>

con este cambio se mejoran los resultados obtenidos. Como podemos ver en el ejemplo de la Figura 51, en el paso de *Negative Mining* vemos que la clasificación de positivo y negativo es bastante más acertada con esta nueva técnica, ya que los negativos 3 y 4 son clasificados de forma errónea con la técnica antigua mientras que la nueva lo hace de forma más acertada.

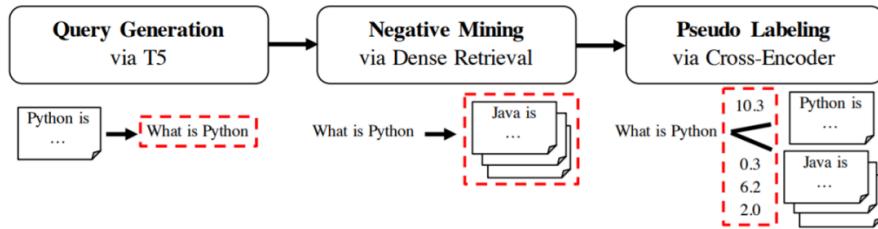


Figura 50. Pasos del método GPL.

Fuente: https://www.sbert.net/examples/domain_adaptation/README.html#gpl-generative-pseudo-labeling

Una vez tenemos las tripletas (*generated query, positive passage, mined negative passage*) y las similitudes calculadas por el *Cross-Encoder* para los pares (*query, positive*) y (*query, negative*) se puede empezar a entrenar al modelo usando *MarginMSELoss*¹ [30] y así poder enseñar al modelo qué textos son realmente relevantes para la consulta dada.

	Item	Text	GPL	QGen
👉 Query asks for definition of "futures contract"	Query	what is futures contract	-	-
	Positive	Futures contracts are a member of a larger class of financial assets called derivatives ...	10.3	1
👉 Easy negatives: Mention "futures contract" only	Negative 1	... Anyway in this one example the s&p 500 futures contract has an "initial margin" of \$19,250, meaning ...	2.0	0
	Negative 2	... but the moment you exercise you must have \$5,940 in a margin account to actually use the futures contract ...	0.3	0
👉 False negative	Negative 3	... a futures contract is simply a contract that requires party A to buy a given amount of a commodity from party B at a specified price...	8.2	0
👉 Hard negative: Give partial definition	Negative 4	... A futures contract commits two parties to a buy/sell of the underlying securities, but ...	6.9	0

Figura 51. Ejemplo GPL vs. QGen.

Fuente: https://www.sbert.net/examples/domain_adaptation/README.html#gpl-generative-pseudo-labeling

¹ https://www.sbert.net/docs/package_reference/losses.html#marginmseloss

3.2.6.4. Conclusiones sobre el entrenamiento

Se han realizado varias pruebas para entrenar estos modelos sobre nuestro conjunto de películas, pero no ha tenido gran impacto en los resultados debido a que estos modelos han sido entrenados con una enorme cantidad de datos de varias fuentes y dominios, como por ejemplo Wikipedia, que ya contiene información de películas, por lo que entrenarlo sobre unos pocos datos más no aporta nada nuevo al modelo.

Además, el lenguaje que se utiliza en las sinopsis no tiene un lenguaje específico, como podría ser el de los textos médicos. Aun así, es relevante conocer la existencia de estas técnicas que resultan muy útiles cuando se quiera adaptar un modelo a un dominio en el que no haya sido entrenado, ya que estos modelos ya “han aprendido el lenguaje” que es la tarea difícil, y solo hay que entrenarlos sobre un conjunto de datos específico (proceso de ajuste fino) para poder adaptarlos a nuestra tarea o dominio sin necesidad de reentrenar un modelo desde cero, lo cual llevaría un gasto en computación y tiempo más que considerable.

3.2.7. Base de datos

Tras obtener los datos del *Cross-Encoder*, toda la información obtenida guardada en un archivo CSV¹ (del inglés *Comma-Separated Values*), es almacenada en la base de datos. Inicialmente se realizó el vaciado de información a la base de datos escribiendo un fichero de texto con todas las instrucciones *INSERT*. Esta opción fue descartada rápidamente por la cantidad de problemas que surgían con los caracteres de escape. Posteriormente se intentó mediante la función del propio *MySQL WorkBench* (gestor de base de datos) importar un CSV indicando los datos que definían el archivo, tales como el carácter de escape, el delimitador de campos, el carácter de encapsulamiento de las cadenas de texto, el carácter de final de línea o el número de filas a ignorar. Esta opción mejoró con creces los tiempos de ejecución, pero tuvimos el mismo problema con los caracteres especiales de algunos títulos o argumentos. Finalmente, se decidió utilizar la biblioteca de Pandas con su instrucción *to_sql*² que permite el vaciado en una base de datos MySQL dado el conector, el archivo CSV, y algunos parámetros de configuración. Con esta opción se resolvieron todos los problemas presentes con los caracteres especiales.

3.2.7.1. Estructura de la base de datos

En las primeras versiones de la aplicación decidimos que lo mejor sería guardar todos los datos en una misma tabla ya que las búsquedas serían únicamente por id y título. Por lo tanto, las consultas tendrían un carácter sencillo. Pero tras la última versión añadimos la búsqueda por actor y la posibilidad de filtrar por género. Para

¹ https://es.wikipedia.org/wiki/Valores_separados_por_comas

² https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_sql.html

la búsqueda por actor decidimos no modificar la estructura de la base de datos y guardar los actores en un campo de tipo *text* de la tabla *Movies* para poder realizar la búsqueda usando nuestro algoritmo. Pero para la búsqueda por género nos vimos con la necesidad de crear una tabla intermedia de tipo $1 - N$ donde una película tiene N géneros. Pero, al realizar las pruebas los tiempos de consultas eran demasiado altos, sobre todo cuando necesitábamos obtener el subconjunto de películas que no tenían a la vez los géneros seleccionados (esta consulta la usábamos para filtrar por género usando el algoritmo). Por esto se nos ocurrió generar una tabla con $1 +$ (el número de géneros totales) como entradas de tal forma que la clave primaria sería el id de la película y el resto de las entradas un conjunto de booleanos que estaban activos si la película era de ese género. De esta forma las consultas pasaron de durar de media 10 segundos a ser prácticamente instantáneas.

La estructura final de la base de datos tiene este diagrama:

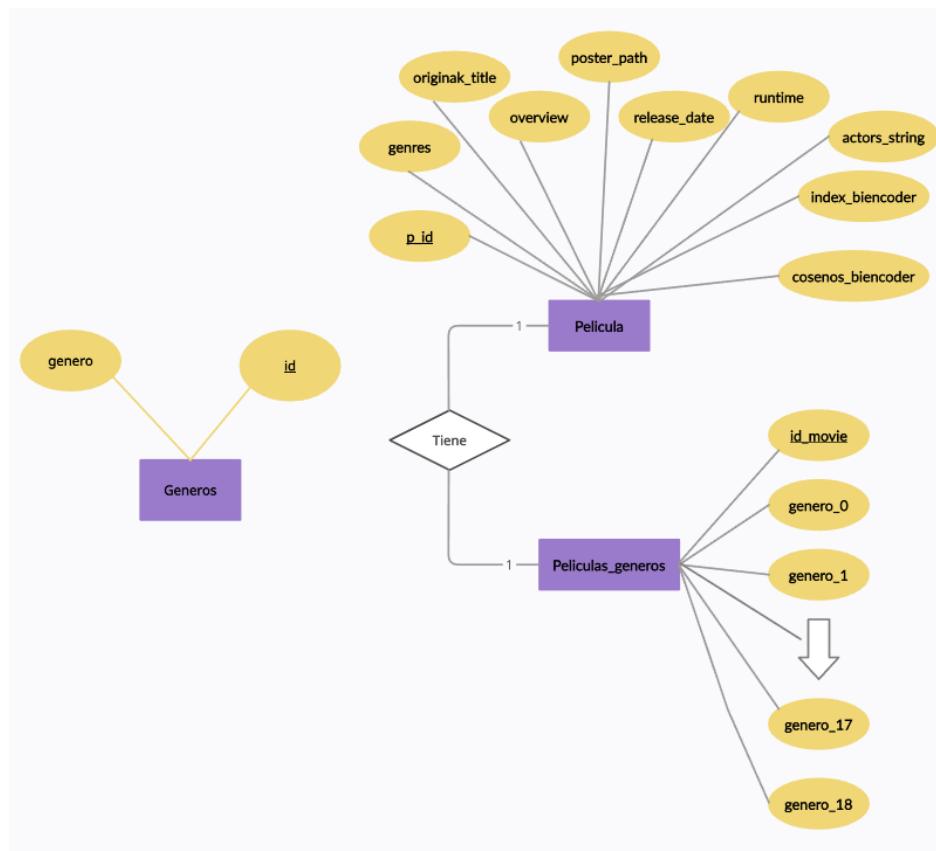


Figura 52. Diagrama de la base de datos.

Una vez almacenada la información en la base de datos hicimos pruebas haciendo diferentes consultas. Se observó que las consultas realizadas por identificador eran bastante rápidas, pero, por el contrario, aquellas relacionadas con el título, la sinopsis y los actores, todos esos campos de tipo *text*, eran significativamente más lentas. Para resolver este problema se decidió indexar la tabla por título, actores y sinopsis usando índices de tipo *fulltext*, lo que aceleró enormemente los tiempos de ejecución de las consultas y mejoró los resultados.

3.2.8. Servidor

El servidor se encarga de conectar la base datos con la interfaz de usuario. Este recoge las peticiones *HTTP*¹ enviadas por la aplicación web y devuelve la información requerida en formato *JSON*². Primero, se usó la clase de *Python http.server*³ que era muy primitiva, pero permitió crear las primeras peticiones y conectar la base de datos al servidor para hacer pruebas en local.

La siguiente mejora surgió buscando distintas bibliotecas que simplificaran el código y permitiesen mayor escalabilidad al proyecto. Para ello se refactorizó el servidor utilizando *Flask*, un entorno de desarrollo de aplicaciones web en *Python*.

El servidor consta de tres partes:

- **Endpoints o puntos de acceso:** reciben las peticiones realizadas por la aplicación web y devuelven una respuesta en formato JSON.
- **El conector con la base de datos:** se encarga de mantener una conexión fluida con la base de datos, realizar peticiones a esta, y transformar los datos a un formato legible por la aplicación web.
- **Algoritmo:** al iniciar el servidor cargamos los *sentence-embeddings* dentro de una clase llamada Algoritmo la cual nos permite hacer cálculos en tiempo real, buscar por sinopsis, título o actores.

3.2.8.1. Puntos de acceso

Todos los datos que necesite la aplicación web son devueltos por los puntos de acceso o *endpoints*. La respuesta que se devuelve es en formato *JSON* y siempre tiene la misma estructura con los siguientes 4 campos:

- **content:** contiene toda la información requerida de la petición. En nuestro caso, actualmente, solo se devuelve un array de películas.
- **state:** devolvemos el estado de la respuesta. Si todo ha ido bien se devolverá una cadena con valor “OK” pero si ocurre algún error se devolverá con valor “FAIL”.
- **cont:** es simplemente el número de películas devueltas en *content*.
- **totalcont:** devuelve la cantidad total de películas que hay guardadas en nuestra base de datos. Como este dato es estático, ya que por ahora no hay

¹ https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto

² <https://es.wikipedia.org/wiki/JSON>

³ <https://docs.python.org/3/library/http.server.html>

forma de insertar películas dinámicamente, *totalcont* se calcula al iniciar el servidor y se trata como una constante.

El servidor consta 7 *endpoints*:

- **GetAllMovies:** es la petición más básica y actualmente no se usa en la aplicación, pero fue necesaria para probar las primeras versiones del servidor. Se realiza la consulta “Select * from movies” a la base de datos. Esta consulta devuelve todas las películas que hay almacenadas, y por último se devuelven en formato *JSON* dentro de la clave *content*.
- **GetMovieById:** pide a la base de datos una película según su id. Además, obtiene las 50 primeras películas más similares según el campo “*index.biencoder*”. Esto se devuelve de esta forma para que la web no tenga que realizar más llamadas y pueda mostrar la información de la película junto con las recomendadas.
- **GetMoviePage:** devuelve las películas paginadas. Para no obtener todas las películas constantemente hemos recurrido al sistema de paginación que nos permite obtener y mostrar 50 películas por página.
- **GetMovieByName:** devuelve las películas según el título que escriba el usuario en la barra de búsqueda y de los géneros que haya seleccionado. Esta petición, aparte de filtrar por los géneros, se puede realizar de 2 formas según la elección del usuario:
 - Realizando la búsqueda como una consulta a la base de datos utilizando los operadores *MATCH* y *AGAINST* que nos permiten una búsqueda eficiente dentro de un texto y devuelve buenos resultados.
 - Realizando el cálculo de similitudes en tiempo real utilizando el algoritmo. De esta forma obtenemos películas donde el título no tiene que coincidir necesariamente y los resultados conseguidos son términos similares a la búsqueda realizada. Por ejemplo, si buscamos la película “Up” los primeros resultados contienen ese texto, pero los últimos referencian a películas cuyo título contiene el texto “Down”.
- **GetMovieByActor:** devuelve las películas según el reparto. Es posible filtrar las películas por géneros y, de forma similar a la petición *GetMovieByName*, se puede realizar de dos formas:
 - Realizando la búsqueda como una consulta a la base de datos y utilizando los operadores *MATCH* y *AGAINST*. Esta búsqueda es aún más eficiente que en la búsqueda por título ya que estos

operadores devuelven un valor numérico de similitud según la coincidencia y repetición de las palabras de más o dos letras. Por ello no tienen por qué estar ordenados los actores. De esta forma es posible buscar un conjunto de actores y se devolverá primero las películas que contengan ambos y después las individuales de cada uno.

- Realizando el cálculo de similitudes en tiempo real utilizando el algoritmo. Esta forma es bastante más impredecible. El resultado, generalmente suelen aparecer las películas individuales de cada actor por lo que consideramos que estos resultados son peores.
- **GetMoviesByText:** una de las implementaciones más importantes de la aplicación es poder obtener las películas más similares según el resumen de las películas y un texto. Esta consulta pide a la clase Algoritmo que calcule las similitudes y devuelva a la aplicación web las películas con los argumentos más similares. Al igual que las dos anteriores, permite el filtrado por género.
- **GetGenresList:** devuelve la lista de géneros y sus *id* para que la web pueda usarla en el filtro de géneros.

Es importante conocer cómo filtramos por géneros cuando se realiza alguna petición al servidor que deba ejecutar el algoritmo. Lo primero es entender que el algoritmo siempre devuelve 50 películas. De este modo, para filtrar por género, a aquellas que no concuerdan con los géneros seleccionados se les asigna una similitud nula provocando que el algoritmo no las recupere. En caso de que no existan 50 películas con los géneros seleccionados, la función escogerá películas aleatorias hasta completar las 50. Por ello, realizamos una segunda comprobación en la que eliminamos todas aquellas que resulten ser no relevantes.

3.2.8.2. Conector con la base de datos

El conector con la base de datos realiza labores similares a las de un objeto DAO¹, es decir, se encargan de suministrar una interfaz entre la aplicación del servidor y los datos devueltos por las consultas que realizamos. En nuestro caso no usamos un objeto si no que hemos preferido implementar un conjunto de funciones que transforman los datos devueltos por la base de datos, en forma de array de tuplas a los tipos que estemos solicitando según la consulta. Actualmente hay tres tipos de consultas según nuestras necesidades:

- **QueryNumber:** transforma el array de tuplas a un dato de tipo *int*. Lo usamos para devolver la cantidad de películas que tiene la base de datos.

¹ https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos

- **QueryGenre**: transforma el array de tuplas a un array de *Genres*, una clase auxiliar creada por nosotros para devolver los géneros a la aplicación web.
- **QueryMovie**: transforma el array de tuplas a un array de tipo *Movie*. Esta clase contiene toda la información de las películas. Esta es la más utilizada en toda la aplicación.

3.2.8.3. Ejecución del algoritmo en tiempo real

La ejecución del algoritmo en tiempo real nos permite poder realizar una búsqueda por argumento, título o actores. De este modo, si se recuerda una película de la cual no sabes su nombre, se puede intentar describirla para que el algoritmo encuentre similares a dicha descripción.

Para que la ejecución del algoritmo en tiempo real sea factible en tiempos y no ralentice mucho la interfaz de usuario, se han realizado varias acciones:

- Se tienen calculados los *sentence-embeddings* de los argumentos, los títulos y los actores y estos son cargados al inicializar el servidor.
- Se utiliza un modelo rápido, pero a su vez bastante exacto como es el *all-MiniLM-L6-v2*.
- Se ejecuta en un entorno que dispone de GPU, lo cual acelera mucho el cálculo. Sin embargo, también es posible ejecutarlo en CPU a costa de un mayor tiempo de ejecución.

Obtener las similitudes entre todas las películas tiene un coste computacional elevado, pero calcular la similitud de una de ellas con el resto es prácticamente instantáneo, esto nos permite poder ejecutar el algoritmo sin interferir en la experiencia del usuario con la interfaz.

3.2.9. Interfaz de usuario

Para una representación visual de la colección de datos y una exploración intuitiva de la misma, se decidió al inicio del proyecto la creación de una interfaz de usuario en forma de aplicación web que permitiese la navegación y la búsqueda de los distintos datos de manera parecida a como se haría en páginas web actuales. Se cogieron como inspiración páginas que actúan también sobre otras colecciones de

datos como IMDB, TMDB, Netflix¹, o Amazon², entre otras. Así mismo, también se decidió que el idioma de la web fuese el inglés.

Por esto, y dado el amplio alcance esperado para la aplicación, la intención es que la exploración de la colección de datos se pueda realizar en un entorno viable para cualquier tipo de usuario, independientemente de los conocimientos de informática que este posea.

3.2.9.1. Implementación

Para la implementación de la lógica de la aplicación web se ha utilizado la biblioteca de *JavaScript*, *React*, dada la comodidad que ofrece a la hora de crear componentes funcionales, o de utilizar otros ya creados por los usuarios.

Por otro lado, para la parte de diseño se han utilizado las hojas de estilo en cascada, *CSS*. Durante el desarrollo de la interfaz se realizaron bocetos de distintas versiones de esta creados en *Balsamiq*³. Estos bocetos se pueden observar en la Figura 53 y la Figura 54 para la versión 0; en la Figura 55 y la Figura 56 para la versión 1; en la Figura 57, en la Figura 58, y en la Figura 59 para la versión 2; y en la Figura 60, en la Figura 61, y en la Figura 62 para la versión 3.

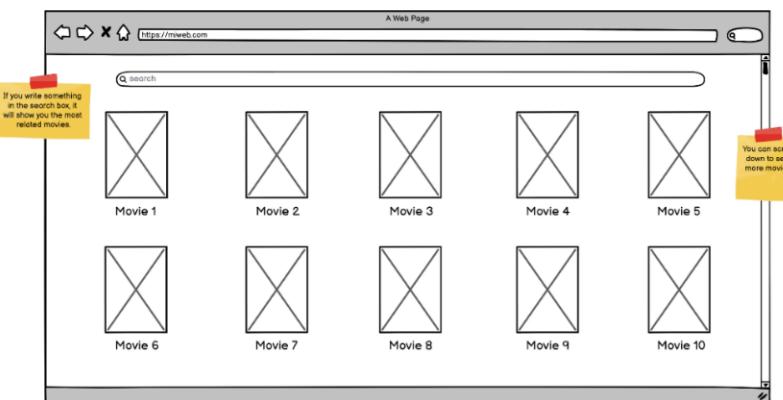


Figura 53. Versión 0, página principal.

¹ <https://www.netflix.com/es/>

² <https://www.amazon.es/>

³ <https://balsamiq.com/docs/>

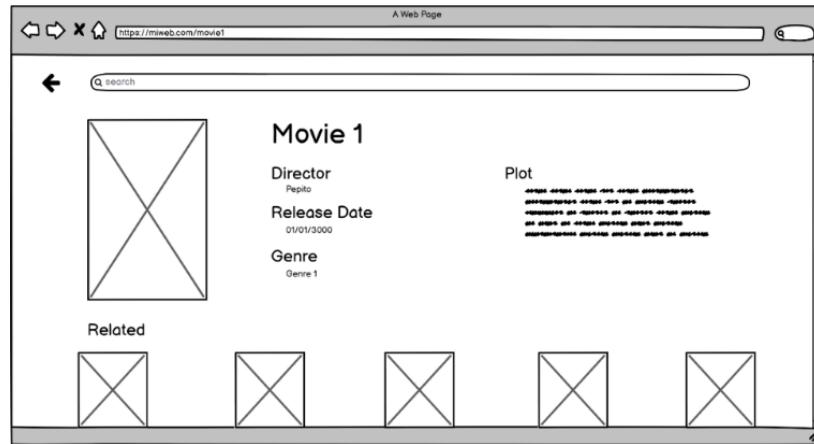


Figura 54. Versión 0, vista detallada de una película.

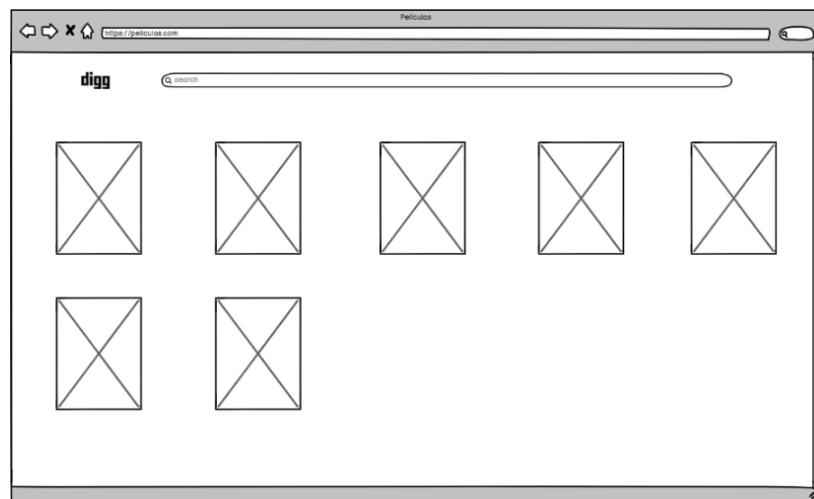


Figura 55. Versión 1, página principal.

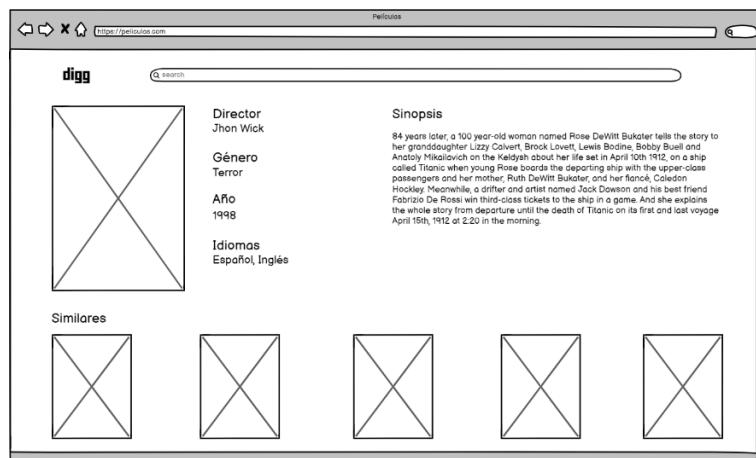


Figura 56. Versión 1, vista detallada de una película.

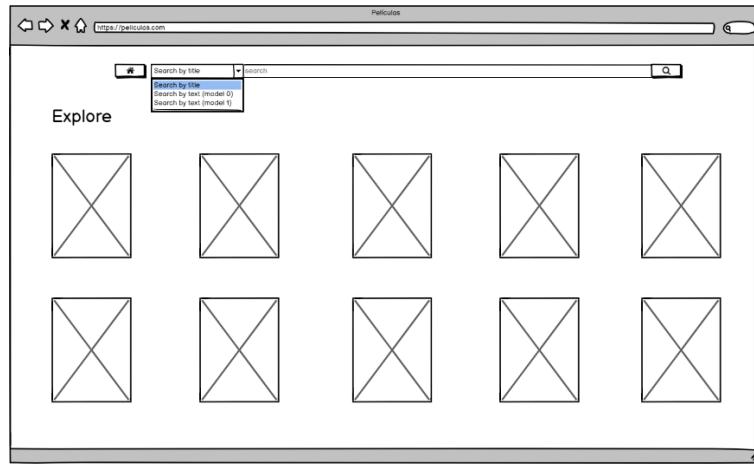


Figura 57. Versión 2, página principal.

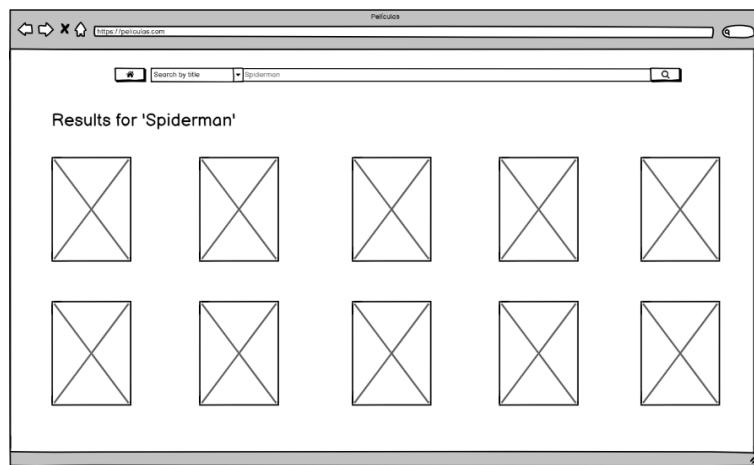


Figura 58. Versión 2, búsqueda de una película.

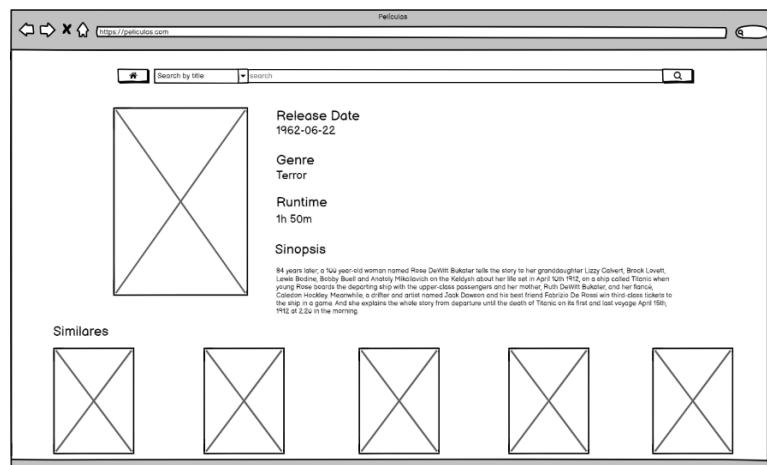


Figura 59. Versión 2, vista detallada de una película.

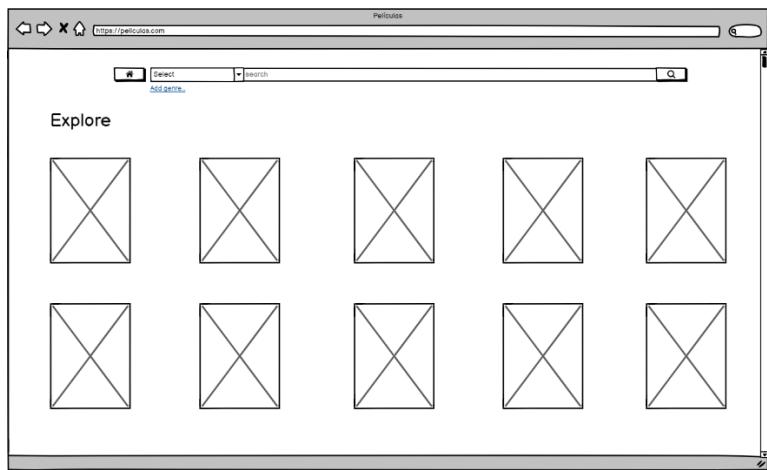


Figura 60. Versión 3, página principal.

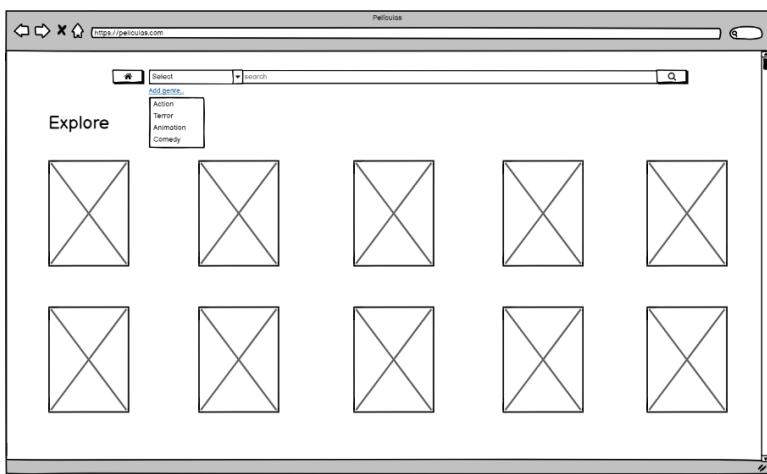


Figura 61. Versión 3, filtro de géneros.

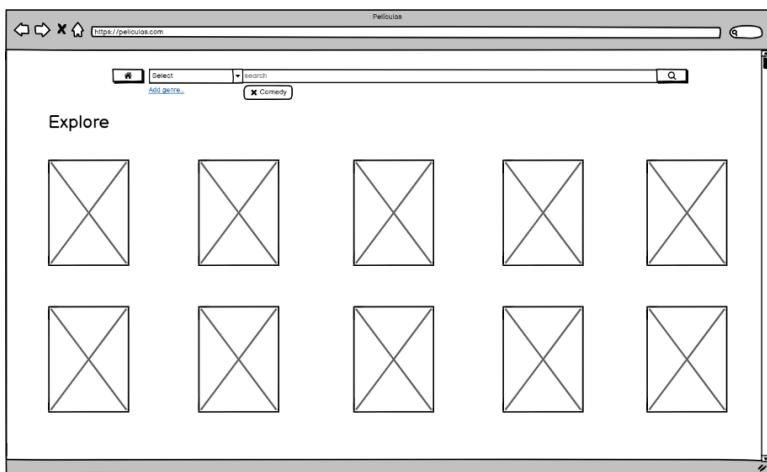


Figura 62. Versión 3, filtro de género seleccionado.

Para una fácil colaboración y distribución de las tareas entre los encargados de la creación de esta aplicación se ha utilizado la plataforma GitHub. Se pueden distinguir así dos repositorios distintos:

- Aquel que contiene todas las iteraciones correspondientes a la primera versión de la aplicación, la cual se caracteriza por ser estática al hacer uso de un archivo CSV para la obtención de los datos generados por el algoritmo¹.
- Aquel que contiene todas las iteraciones correspondientes a una segunda versión de la aplicación, considerada la versión final, en la que se han utilizado llamadas a la base de datos para la obtención de los datos generados por el algoritmo².

Gracias al servicio GitHub Pages³ se pudo publicar de manera gratuita la primera versión de la aplicación web, la cual se puede visitar desde el siguiente enlace:

<https://rafanoble.github.io/react-website/>

Sin embargo, dada la naturaleza dinámica de la segunda versión de la aplicación se requirió la creación de un servidor local para el acceso a la base de datos desde la aplicación. Por este motivo, la segunda versión se deberá ejecutar localmente siempre que se quiera acceder a ella.

3.2.9.2. Estructura de la interfaz

A continuación, se pasa a enumerar las distintas partes que componen la versión final de la aplicación web:

- 1) **Cuerpo de la aplicación, <App>**: es el componente base que contiene toda la lógica de la aplicación y el encargado del renderizado de todas las pantallas y componentes de esta.
- 2) **Pantallas**: todas ellas tienen en común la barra superior de navegación que permite el regreso a la pantalla principal <Homepage> y la búsqueda de datos en cualquier momento de la ejecución.
 - a) <Homepage>: es la pantalla principal de la aplicación. En ella se muestra un listado con todas las películas almacenadas en la base de datos y los botones de navegación entre páginas. En dicha pantalla se realiza una consulta a la base de datos para obtener el listado de las películas de la página en la que se encuentre el usuario actualmente.

¹ <https://github.com/lPoloBlash/tfg-web>

² <https://github.com/RafaNoble/react-website>

³ <https://pages.github.com/>

- b) <Movie>: es la pantalla que muestra los detalles de una película seleccionada. Seguidamente, aparece un listado de las cincuenta primeras películas similares a la seleccionada. En dicha pantalla se realiza una consulta para obtener los datos de la película seleccionada y un listado de las cincuenta primeras películas similares.
 - c) <SearchResults>: pantalla en la que se muestran los resultados de cualquier búsqueda realizada sobre la colección de datos usando la barra superior de navegación. Su estructura es igual a la de <Homepage>. En dicha pantalla se realiza una consulta a la base de datos con los datos introducidos por el usuario en la barra de búsqueda para obtener un listado con las películas resultantes de dicha búsqueda.
 - d) <NotFound>: pantalla auxiliar utilizada para redirigir al usuario en caso de acceso a una dirección no existente dentro del dominio de la aplicación. Muestra un mensaje de error informando de que la página a la que se está intentando acceder no existe (error 404¹) e instando al usuario a volver a la página principal <Homepage> usando la barra superior de navegación.
- 3) **Componentes:** en general, agrupan otros componentes más básicos para facilitar la reutilización de estos.
- a) <HomeButton>: botón encargado de redirigir al usuario a la pantalla principal <Homepage>.
 - b) <earchBar>: este componente es el encargado de renderizar la barra y el botón de búsqueda.
 - c) <GenreFilter>: componente con el que el usuario puede elegir los géneros con los que se filtrarán las películas resultantes de una búsqueda.
 - d) <TopBar>: agrupa los componentes de <HomeButton>, <earchBar>, <GenreFilter> y un cuarto componente <input> que sirve como botón para elegir la forma en que se realiza la búsqueda, ya sea mediante una consulta a la base de datos, o mediante el algoritmo. Contiene además un quinto componente, <Dropdown>, que renderiza un desplegable con tres opciones distintas de búsqueda:
 - i) Búsqueda por título, que indicará al componente <SearchResults> que la consulta de búsqueda a realizar será del tipo búsqueda por título de la película. Se puede realizar mediante una consulta a la base datos o mediante el algoritmo.

¹ https://es.wikipedia.org/wiki/HTTP_404

- ii) Búsqueda por actores, que indicará al componente <SearchResults> que la consulta de búsqueda a realizar será del tipo búsqueda por actores pertenecientes al reparto de una película. Se puede realizar mediante una consulta a la base datos o mediante el algoritmo.
 - iii) Búsqueda por texto, que indicará al componente <SearchResults> que la consulta de búsqueda a realizar será del tipo búsqueda por texto, calculando el algoritmo la similitud del texto buscado con la descripción de las películas. Esta búsqueda solo se puede realizar mediante el algoritmo.
- e) <MovieBox>: este componente se encarga de renderizar un contenedor que mostrará el póster de la película, y debajo, el título de esta. Es el componente base de cualquier listado de películas.
- f) <MovieList>: este componente renderiza un listado de películas usando <MovieBox> como componente base.
- g) <MovieInfo>: componente encargado de renderizar un contenedor con los detalles de una película seleccionada. Estos detalles son: el título original, el póster, la fecha de estreno (en formato aaaa/mm/dd), el/los género/s, la duración (en horas y minutos) y un resumen de la película.
- h) <PageButtons>: componente que agrupa el renderizado de los botones y la lógica para la paginación de los listados.
- i) <ScrollToTop>: componente auxiliar que reinicia la posición de la ventana del navegador cada vez que el usuario cambia de pantalla.
- 4) **Diseño:** cada pantalla y componente tiene su CSS complementario en el que se define el diseño que tendrán al renderizarse en la ventana del navegador. Estos archivos contienen toda la información relevante para indicar la distribución de los componentes en pantalla, su forma, su color, e incluso ciertas animaciones.
- 5) **Recursos adicionales:** un archivo que contiene constantes que se usan en las demás partes de la aplicación y una imagen auxiliar para sustituir el póster de aquellas películas que no poseen uno.

Además de las versiones mencionadas con anterioridad, también se llevó a cabo la implementación de una segunda aplicación en Java usando el entorno de desarrollo de Spring¹. Esta aplicación fue creada en paralelo junto con la aplicación final debido a la necesidad por parte del equipo encargado del desarrollo del algoritmo de tener una aplicación más sencilla y plenamente funcional para la depuración y la

¹ <https://spring.io/>

realización de pruebas. Una vez que la lógica de la aplicación final fue terminada e integrada con la base de datos, pasó utilizarse esta para la realización de las pruebas.

3.2.9.3. Funcionalidades

Las funcionalidades implementadas en la aplicación web son las siguientes:

- **Vista de películas sugeridas con paginación:** pantalla principal donde se muestran todas las películas paginadas.
- **Vista de información de una película y sus similares:** donde, al seleccionar una película, se mostrará su información destacada y una lista de cincuenta películas ordenadas según su similitud con ella.
- **Búsqueda de películas por título mediante consulta a la base de datos:** realiza la búsqueda de las películas que contengan la palabra buscada en su título mediante una consulta a la base de datos.
- **Búsqueda de películas por título mediante el algoritmo:** realiza la búsqueda de las películas que contengan la palabra buscada en su título mediante el algoritmo.
- **Búsqueda de películas por texto:** realiza la búsqueda de las películas calculando la similitud entre su descripción y el texto buscado.
- **Búsqueda de películas por actores mediante consulta a la base de datos:** realiza la búsqueda de películas que tengan en su reparto el actor/los actores escritos en la barra de búsqueda mediante una consulta a la base de datos.
- **Búsqueda de películas por actores mediante el algoritmo:** realiza la búsqueda de películas que tengan en su reparto el actor/los actores escritos en la barra de búsqueda mediante el algoritmo.
- **Búsqueda de películas con filtros de género:** realiza la búsqueda de las películas filtrando según los géneros seleccionados. Pueden emplearse solo los filtros en una búsqueda o en conjunto con un tipo de búsqueda concreto.
- **Regreso a la página principal:** mediante un botón situado en la barra superior de navegación.

Por el momento, estas son todas las funcionalidades de las que consta la aplicación. Sin embargo, esta lista podría ampliarse en un futuro con otras como añadir otros tipos de filtros en la búsqueda, ordenación de las películas por orden alfabético, fecha de estreno, popularidad, etc., o incluso la posibilidad de que el usuario pueda añadir más películas.

3.2.10. Ejemplos de uso

- **Vista de películas sugeridas con paginación:**

En la página principal hay una vista de todas las películas disponibles paginadas, con distintos botones para volver a la página principal, para cambiar el tipo de búsqueda, para buscar respecto al contenido dentro de la barra de búsqueda, ya sea usando filtros de género o no y, además, ofrece la opción de ejecutar la búsqueda mediante una consulta a la base de datos o utilizando el algoritmo. En la Figura 63 se puede observar la estructura de dicha pantalla.

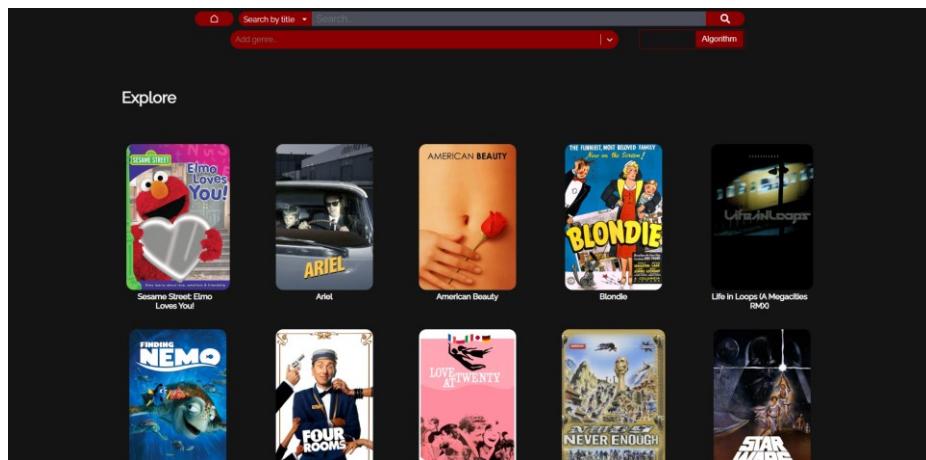


Figura 63. Vista de películas sugeridas con paginación.

Como se puede ver en la Figura 64, al final de la pantalla hay dos botones de paginación. Al pulsar el botón derecho se avanza de página, siempre que no se esté en la última, y al pulsar el botón izquierdo se retrocede, siempre que no se esté en la primera página:

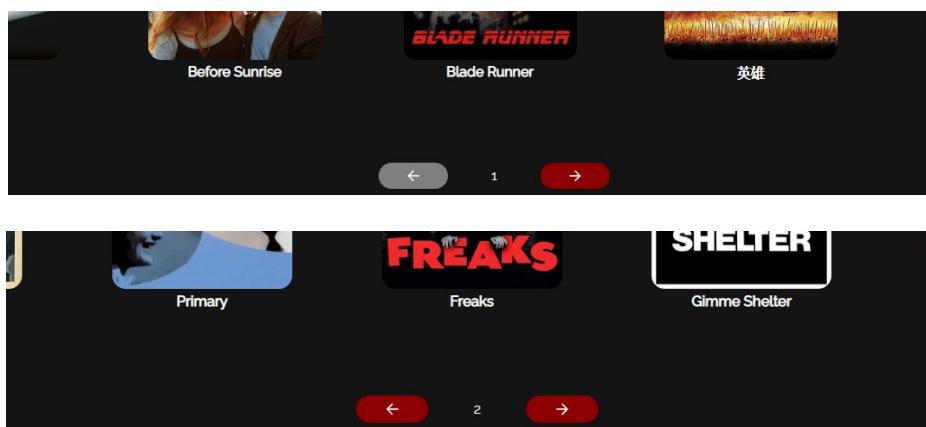


Figura 64. Botones de paginación.

- **Vista de información de una película y sus similares:**

Al hacer clic sobre una película, ya se ubique ésta en la pantalla principal o en la lista de películas relacionadas de otra, redirige al usuario a otra pantalla donde se muestra la información de la película seleccionada (título original, fecha de estreno, género/os, duración, y resumen) y una lista de películas similares a esta. Esta página se puede observar en la Figura 65.

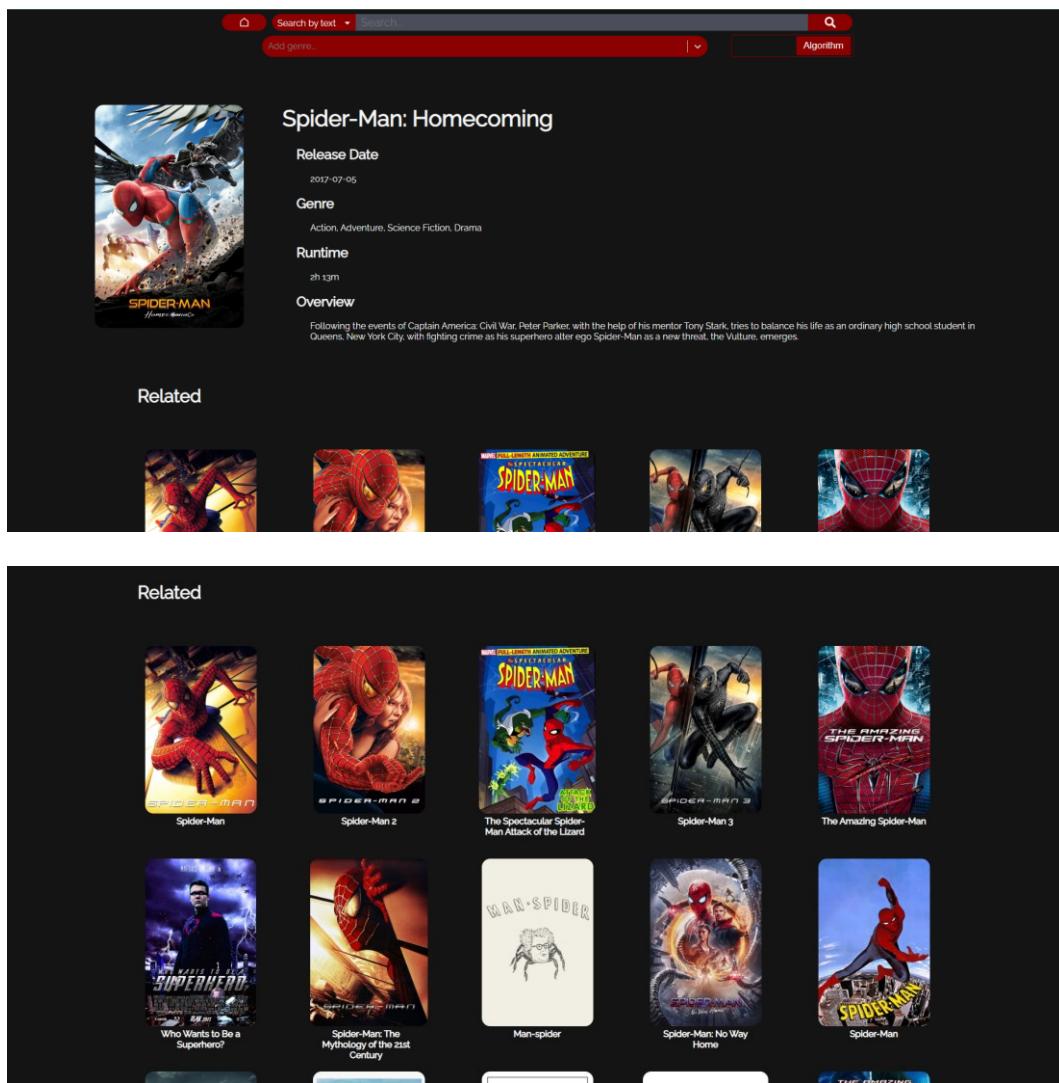


Figura 65. Vista de información de una película y sus similares.

- **Búsqueda de películas por título mediante consulta a la base de datos:**

Al realizar una búsqueda se pueden cambiar diversos parámetros para alterar los resultados de esta. Mediante un desplegable ubicado a la izquierda de la barra de búsqueda se puede cambiar el tipo de búsqueda a “Search by title”. Adicionalmente, se puede elegir si realizar dicha búsqueda mediante una consulta a la base de datos (opción “Query” en el botón situado abajo a la derecha de la barra superior de navegación) o mediante el algoritmo (opción “Algorithm” en el mismo botón). Al acceder por primera vez a la aplicación, la opción por defecto para el tipo de búsqueda será “Search by title” mediante una consulta a la base de datos. En la Figura 66 se puede observar el comportamiento por defecto.

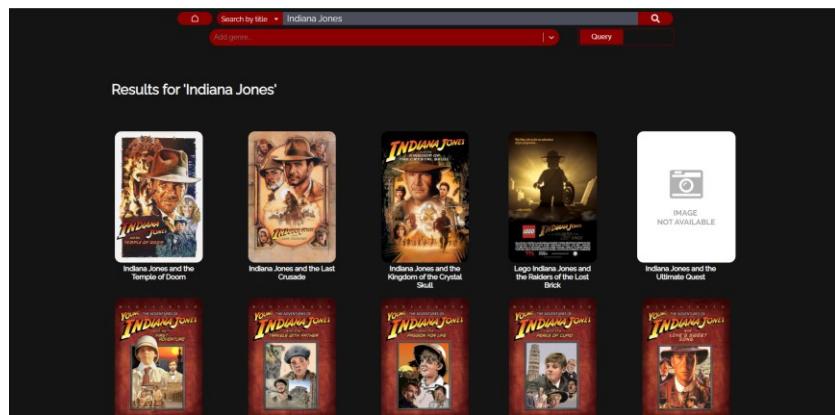


Figura 66. Búsqueda de películas por título mediante consulta.

- **Búsqueda de películas por título mediante el algoritmo:**

Si seleccionamos en el desplegable anteriormente descrito la opción “Algorithm” podremos realizar la búsqueda de películas por título empleando el algoritmo. En la Figura 67 se puede observar dicha opción de búsqueda.

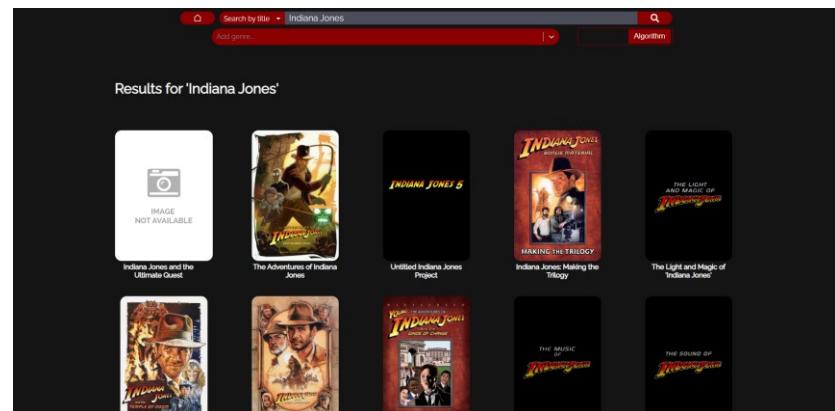


Figura 67. Búsqueda de películas por título mediante el algoritmo.

- **Búsqueda de películas por actores mediante consulta a la base de datos:**

Otra opción disponible a la hora de buscar películas es hacerlo en función de los actores que aparecen en ellas. Para ello, seleccionamos en el desplegable la opción “*Search by actor*”. En la Figura 68 podemos observar la opción mediante consulta a la base de datos.

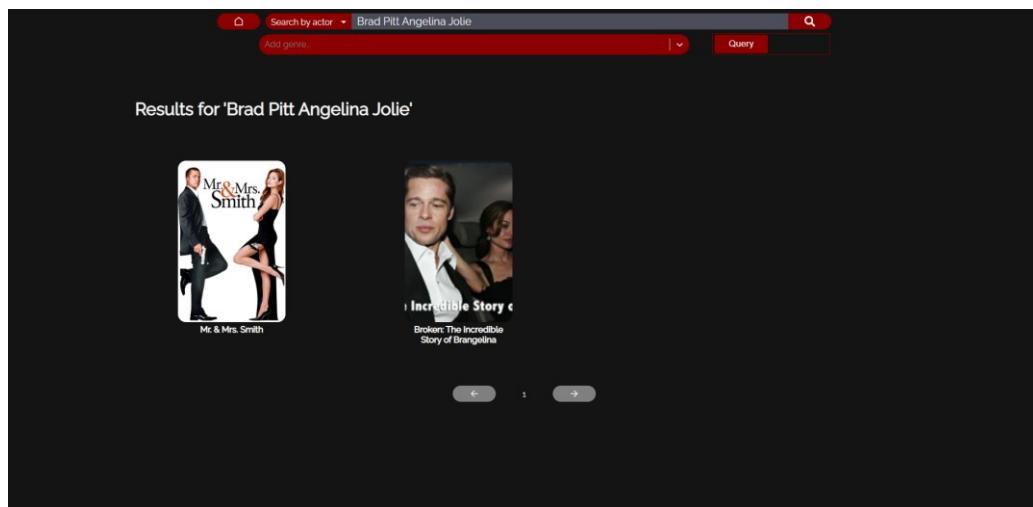


Figura 68. Búsqueda de películas por actores mediante consulta.

- **Búsqueda de películas por actores mediante el algoritmo:**

Al igual que en otras opciones de búsqueda, también podemos seleccionar la opción de búsqueda mediante el algoritmo para la búsqueda de películas por actores. En la Figura 69 podemos ver dicha opción.

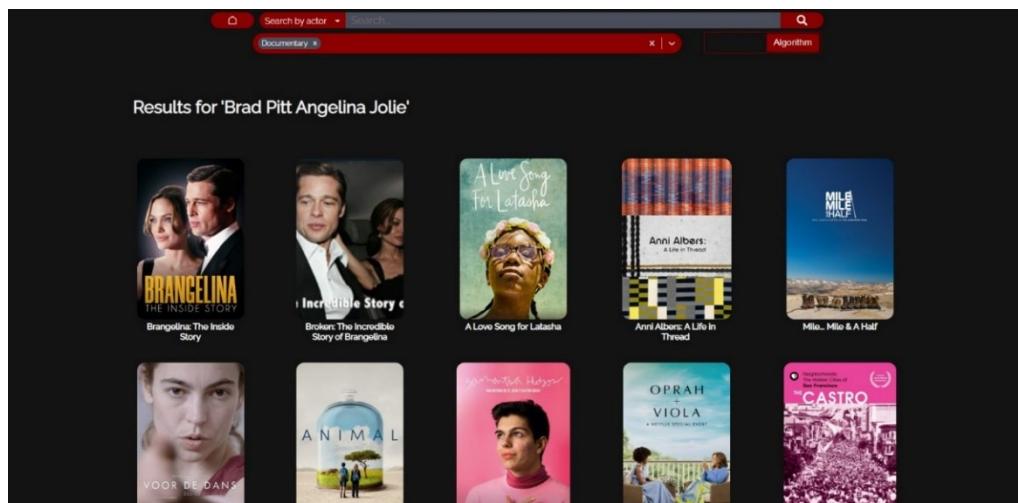


Figura 69. Búsqueda de películas por actores mediante el algoritmo.

- **Búsqueda de películas por texto:**

Accediendo al desplegable podemos cambiar el tipo de búsqueda a “*Search by text*” para buscar películas en función de la similitud existente entre el texto introducido en la barra de búsqueda y el resumen de las películas. Para esta opción solo se puede ejecutar la búsqueda empleando el algoritmo, como se puede observar en la Figura 70.

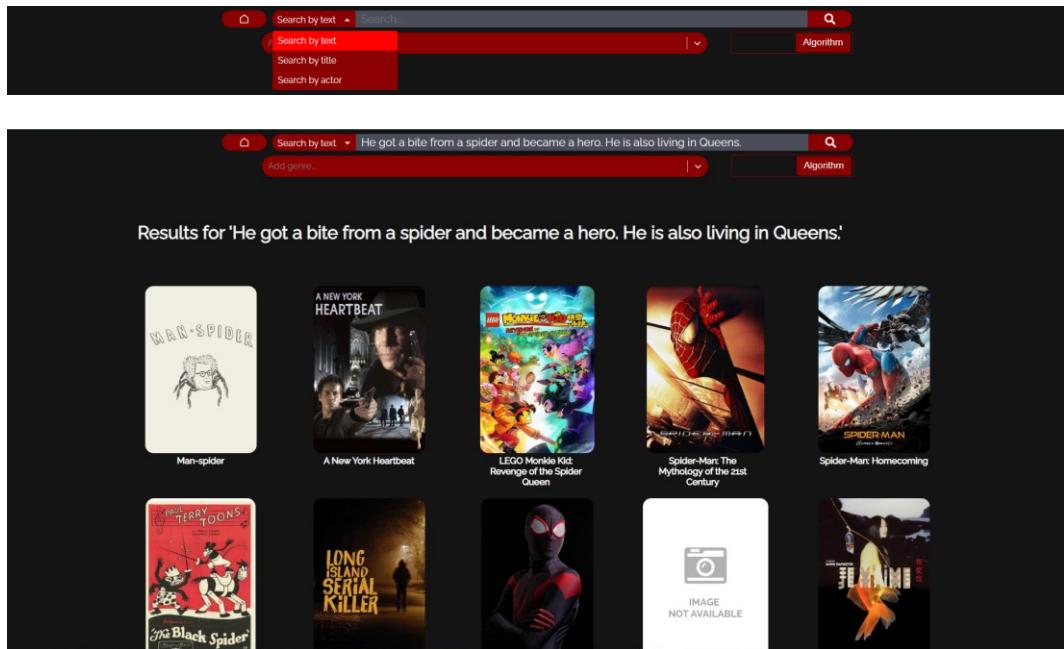


Figura 70. Búsqueda de películas por texto.

- **Búsqueda de películas con filtros de género:**

Podemos también realizar una búsqueda filtrando por los géneros a los que pertenecen las películas. Debajo de la barra de búsqueda encontramos otro desplegable con el que podremos elegir por qué géneros queremos filtrar. Además, también podremos deseleccionar los géneros que hayamos seleccionado previamente uno a uno o todos a la vez. Una vez seleccionados los géneros por los que querremos filtrar, solo tendremos que pulsar el botón de búsqueda para obtener los resultados de la filtración. Además de los géneros, también podremos acotar aún más nuestra búsqueda combinando los filtros con cualquier palabra introducida en la barra de búsqueda para cualquier tipo que hayamos seleccionado (por título, por actores, o por texto). Podemos observar esta funcionalidad en la Figura 71.

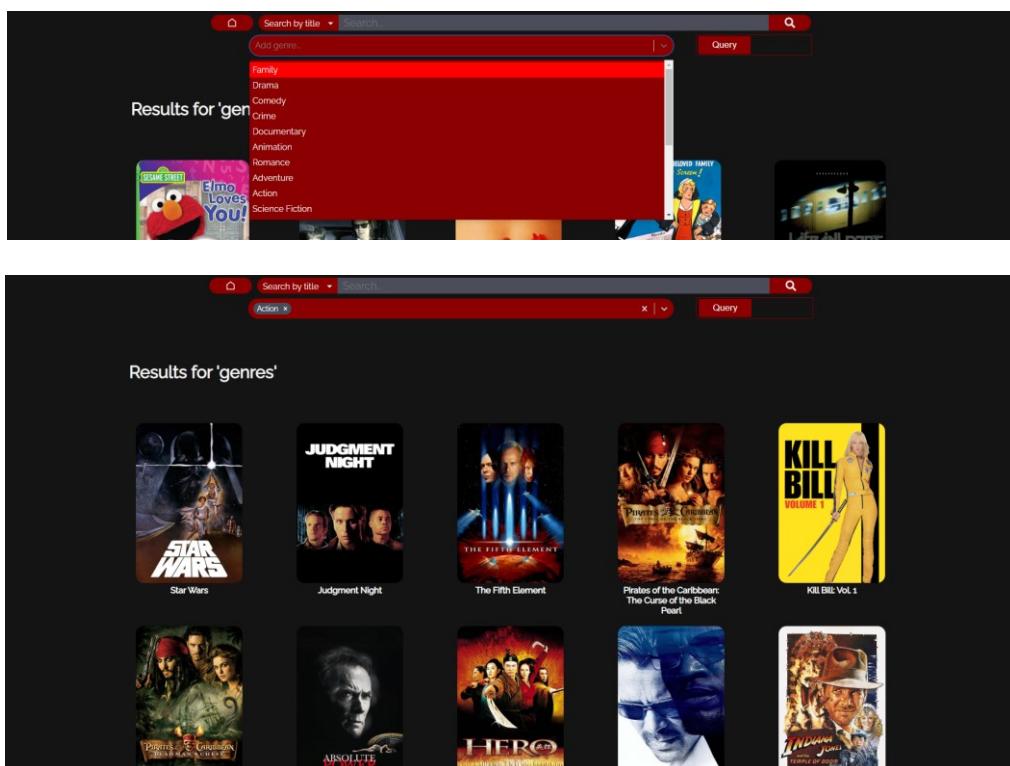


Figura 71. Búsqueda de películas con filtros de género.

- **Vuelta a la página principal:**

En cualquier momento de nuestra exploración mediante la aplicación, podremos pulsar el botón izquierdo ubicado en la barra superior de navegación que nos redirigirá a la página principal con la lista de películas sugeridas vista anteriormente. Este comportamiento se puede observar en la Figura 72.

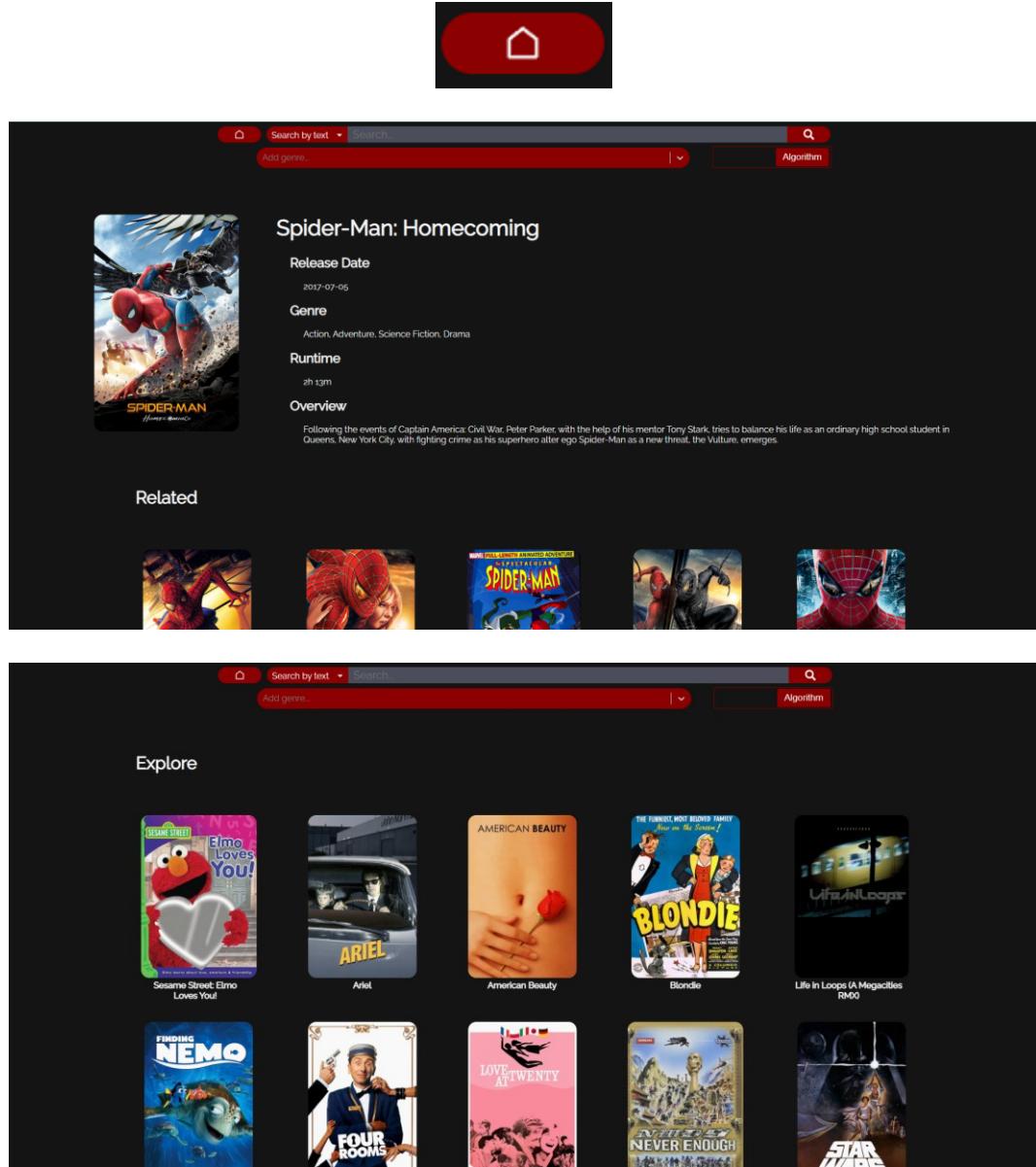


Figura 72. Vuelta a la página principal.

4. Aplicaciones prácticas

En este apartado se recogen los resultados obtenidos al aplicar el SBMC desarrollado sobre diferentes conjuntos de datos, incluyendo el conjunto utilizado para el desarrollo de la web de navegación de películas.

4.1. Exploración de películas

Este conjunto de datos es el mismo que se utiliza en la web del proyecto y que se ha procesado con el objetivo de realizar una exploración en base a la sinopsis de estas.

Uno de los métodos de exploración disponibles es obtener las películas similares a una dada del conjunto. Podemos observar un ejemplo en la Figura 73.

Texto a explorar:

Aragorn is revealed as the heir to the ancient kings as he, Gandalf and the other members of the broken fellowship struggle to save Gondor from Sauron's forces. Meanwhile, Frodo and Sam take the ring closer to the heart of Mordor, the dark lord's realm.

similarity	title	overview
0.72	The Lord of the Rings: The Two Towers	Frodo and Sam are trekking to Mordor to destroy the One Ring of Power while Gimli, Legolas and Aragorn search for the orc-captured Merry and Pippin. All along, nefarious wizard Saruman awaits the Fellowship members at the Orthanc Tower in Isengard.
0.58	Little Nicky	After the lord of darkness decides he will not cede his throne to any of his three sons, the two most powerful of them escape to Earth to create a kingdom for themselves. This action closes the portal filtering sinful souls to Hell and causes Satan to wither away. He must send his most weak but beloved son, Little Nicky, to Earth to return his brothers to Hell.
0.57	Beastmaster III: The Eye of Braxus	Before he died, Dar's father gave a mysterious amulet to Tal, Dar's younger brother who is now king. Dar, while wandering with his animal companions, chances to meet and rescue a family who seek the help of King Tal against Lord Agon, a sorcerer who has conquered their land. Dar obtains an audience for them with Tal, who rallies his troops to march against Agon in the morning. Alas, the young king is captured by Agon's crimson warriors during the night. It is the amulet -- the Eye of the imprisoned god Braxus -- that Agon really wants. But Dar now carries half of it, and is nowhere to be found. Will he fall into Lord Agon's clutches when he comes to free his brother?
0.57	Bal Ganesh 2	Lord Ganesha evolves from a mischievous boy, who is not only innocent but also intelligent, to an elephant-headed god, who embarks on a series of adventures with his friends to defeat evil forces.
0.57	The Dragons of Camelot	The Kingdom of Camelot is plunged into darkness after the death of noble King Arthur, as the reign of his cruel sorceress sister, Morgan Le Fay, begins. Commanding three deadly dragons with her dark magic, she hunts the remaining Knights of the Round Table to the ends of the Earth.

Figura 73. Películas similares en sinopsis a The Lord of the Rings: The Return of the King.

También se pueden recuperar películas con un argumento similar a un texto proporcionado por parámetro. En la Figura 74 y Figura 75 se pueden observar algunos ejemplos.

similarity	title	overview
0.55	Je me souviens	A woman seeks revenge over false accusations of murder.
0.53	Snuff	A troubled young woman, reeling from her parents' deaths, snaps and takes revenge against her tormentors, while two detectives attempt to discover the truth behind her family's frightening past.
0.51	Dama de Noche	A woman who was maltreated during her younger years gets revenge on the people who abused her.
0.51	Defenceless: A Blood Symphony	A woman seeks revenge after being brutally attacked by dishonest land developers.
0.51	நெருசம் மற்புப்பதில்லை	A god-fearing young woman who takes up the job of a caretaker in a rich household tries to take revenge on her lustful employer.
0.51	Girl From Iceland	A mysterious woman seeks revenge against the thugs who abused her and left her for dead.
0.50	Lady Dragon 2	A woman seeks revenge on the men who raped her and murdered her husband.
0.50	Catcall	A woman, fed up with the abuses she suffers while walking on the street, takes her revenge against a catcaller.
0.50	Catcall: Omega Violence	A woman, fed up with the abuses she suffers while walking on the street, takes her revenge against a catcaller.
0.49	Savage Vengeance	A woman is brutally raped by four men, and she plans to seek vengeance. After five years, she hasn't gotten over it yet, and she and her friend are again raped by four men. This time, she tracks them down and finally has her revenge.
0.49	Promising Young Woman	A young woman, traumatized by a tragic event in her past, seeks out vengeance against those who crossed her path.
0.48	The Atonement of Janis Drake	A woman seeks revenge against those who were involved in a sexual assault against her back in college.
0.48	12 Point Kill	Oppressed by her overbearing and violent stepfather, a young woman takes revenge into her own hands.
0.48	Mirror of Death	An abused woman conjures up a spirit to take revenge on the men who have misused her.

Figura 74. Resultados obtenidos con el texto “A woman seeks revenge”.

similarity	title	overview
0.45	CUBE 一度入ったら、最後	Six complete strangers with widely varying personalities are involuntarily placed in an endless maze of interlocking cube-shaped rooms containing deadly traps.
0.45	Who Killed Doc Robbin?	A group of people find themselves trapped in a creepy mansion, complete with secret passageways, a mad doctor and a murderous gorilla.
0.45	Cubed	A group of strangers are trapped inside a massive labyrinth cube and struggle to find their way out. A reboot of the 1997 film, 'Cube'.
0.44	YÖN VARJO	Documentary short with horror elements describes three people experiencing horrifying events, when unknown intruder enters their bedroom at night.
0.43	Storage 24	Something nasty is lurking inside a secure storage unit. When a group of people get trapped inside, they need to find a way to get out of a building that's designed to keep things in...
0.43	Gui wu yan yu	A group of travelers get trapped in a haunted house.
0.43	Cube	A group of strangers find themselves trapped in a maze-like prison. It soon becomes clear that each of them possesses the peculiar skills necessary to escape, if they don't wind up dead first.
0.42	சங்கு சக்கரம்	A group of kids, a couple of kidnappers, and two men who are planning to murder one of the children get trapped inside a haunted mansion.
0.42	Escape: A Ghost Story	A group of escape room employees must face their fears and overcome a series of murderous puzzles set by a malicious ghost who has haunted the building or face the dire consequences.
0.42	Locked in a Room	"Locked in a Room" is a story about three teenagers that are trapped in three separate rooms. They are monitored and watched by a masked man who takes notes on paper as to their every move and emotion. They are further tormented by an eerie voice of someone outside of the room. Throughout, they use their minds to try to think as to why they are there. As time elapses the stakes and chances for survival are slim, but the question remains: who is behind this - the end reveals a startling answer.
0.42	蟄伏	In a locked room where three men were tied up, in order to escape, they have to solve the mystery behind all this.
0.42	Curral de Mulheres	Young women in the Amazon are kidnapped by a ring of devil-worshipers, who plan to sell them as sex slaves. Some of the women escape, but are pursued into the jungle by their captors. The women must band together to turn the tables on their kidnappers.

Figura 75. Resultados obtenidos con el texto “People locked in a cube in which there are traps”.

Además de utilizar la herramienta para comparar argumentos, esta también se utiliza en la web para la búsqueda por título y por actores. Se pueden ver diferentes ejemplos de los resultados obtenidos para diferentes búsquedas de títulos en la Figura 76, Figura 77 y Figura 78. En la Figura 79 se puede ver un ejemplo de búsqueda por actores.

Los resultados obtenidos son bastante acertados incluso sin escribir de la forma exacta en la que aparece en los datos el texto a buscar.

similarity	original_title
0.56	Spider-Man
0.56	Spider-Man
0.56	Spider-Man
0.54	Man-spider
0.54	Spider-Man: The Ultimate Villain Showdown
0.53	Spider-Man 2
0.52	Spiderman, Save Me
0.52	Daredevil vs. Spider-Man
0.51	Spider
0.51	Super Spider
0.51	The Amazing Spider-Man
0.51	The Spider
0.50	Spider-Man: Into the Spider-Verse
0.50	Spider-Man 3
0.49	The Amazing Spider-Man 2
0.49	Spider-Man: Across the Spider-Verse
0.49	Spider-Man: Homecoming
0.49	Spider-Man: Beyond the Spider-Verse
0.49	Viva Spider-Man
0.48	Italian Spiderman
0.48	Spider-Man: Convergence
0.48	Spider-Man: The Mythology of the 21st Century
0.48	Spider-Man 2: Making the Amazing
0.48	The Spider Woman

Figura 76. Resultados obtenidos con el texto “Spiderman”.

similarity	original_title
0.50	J.R.R. Tolkien: Master of the Rings - The Definitive Guide to the World of the Rings
0.48	The Lord of the Rings
0.46	The Trilogy
0.46	The Trilogy
0.46	The Lord of the Rings: The Two Towers
0.45	J.R.R. Tolkien: The Origin Of The Rings
0.45	The Lord of the Rings: The Fellowship of the Ring
0.45	Bugs: A Trilogy
0.44	Trilogy
0.44	Trilogy
0.44	Movieland Magic
0.44	Georges Marvelous Trilogy
0.44	The Lord of the Rings: The Return of the King
0.44	J.R.R. Tolkien and the Birth Of "The Lord of the Rings" And "The Hobbit"
0.44	Fantastic Beasts: The Secrets of Dumbledore
0.43	Dank Mysteries

Figura 77. Resultados obtenidos con el texto “Lord of rings”.

similarity	original_title
0.49	Star Wars
0.48	The Stars of Star Wars
0.47	Star Film
0.46	Star Wars: Greatest Moments
0.46	Star Wars: Heroes & Villains
0.46	The Mythology of Star Wars
0.45	The Secret of Star Wars
0.45	Star Wars Begins: A Filmmumentary
0.45	The Making of Star Wars
0.45	Love Wars
0.44	The Story of Star Wars
0.44	Battle Star Wars
0.44	Night Wars
0.44	Video Wars
0.44	The Characters of Star Wars
0.44	Stars and Strife
0.44	Star Wars: The Clone Wars - The Nightsisters Trilogy
0.43	Star Wars Biomes
0.43	Star Wars: The High Republic
0.43	Star Wars Nothing But Star Wars
0.42	Film Star
0.42	The Wars
0.42	The Wars
0.42	Star Wars: The Old Republic
0.42	Sandwich Wars
0.42	Star Wars: Ewoks - Tales from the Endor Woods
0.42	Fox Wars
0.42	Star Wars: In Concert
0.42	From Star Wars to Star Wars: The Story of Industrial Light & Magic
0.42	Star Wars: Rise of Skywalker Documentary

Figura 78. Resultados obtenidos con el texto “Starwars”.

similarity	title	actors_string
1.00	Broken: The Incredible Story of Brangelina	Brad Pitt,Angelina Jolie
0.99	Brangelina: The Inside Story	Angelina Jolie,Brad Pitt
0.87	Formula One	Brad Pitt
0.87	He Wanted the Moon	Brad Pitt
0.87	Brad Pitt in America's National Parks	Brad Pitt
0.83	Angela & Viril	Angelina Jolie
0.83	Alice & Viril	Angelina Jolie
0.81	Untitled George Clooney/Brad Pitt/Jon Watts Project	Brad Pitt,George Clooney
0.67	Every Note Played	Christoph Waltz,Angelina Jolie

Figura 79. Resultados obtenidos con el texto “Brad Pitt, Angelina Jolie”.

Igualmente se han realizado algunas pruebas con las sinopsis en español obtenidas de la web de TMDB durante la extracción de datos. En este caso se usará el modelo *paraphrase-multilingual-MiniLM-L12-v2* para calcular los *sentence-embeddings*. El número de películas en este caso es menor, rondando los 100.000 elementos. Como se puede observar en la Figura 80 y Figura 81 los resultados obtenidos al realizar una exploración en tiempo real con un texto proporcionado siguen siendo acertados gracias al uso de un modelo multilenguaje.

Texto a explorar:

trata de personas encerradas en un cubo que tienen que escapar

similarity	title	overview_es
0.77	CUBE 一度 入ったら、最後	Un cubo misterioso. Seis hombres y mujeres se ven de repente atrapados en 'Cube'. No está claro dónde está, ni por qué están atrapados, ni si hay una salida, ni si sobrevivirán, ni siquiera qué es la habitación.
0.71	Cube 2: Hypercube	Ocho personas despiertan encerradas en una extraña habitación con forma de cubo sin saber cómo llegaron hasta ahí. Pronto descubren que las leyes de la física no se aplican en ese extraño mundo, así que tendrán que descubrir los secretos del cubo para poder sobrevivir.
0.68	Star Runners	Un grupo de personas son elegidas por el Gobierno para realizar una misión secreta, en la que tendrán que recuperar una "caja" de un extraño y lejano planeta. Al darse cuenta de que la caja contiene a una persona, escapan en un transbordador que resulta ser atacado y posteriormente se estrella en un planeta en donde habitan seres extraños en donde deberán escapar de allí con vida.
0.68	The Box	La historia de una caja llena de seres miserables y un espécimen que trata de librarse del sino de sus vecinos.
0.68	Circle	Cincuenta desconocidos se despiertan encarcelados juntos dentro de una cámara misteriosa formando un círculo. En tiempo real, deben identificar a sus captores y decidir cuál de ellos merece sobrevivir, en un juego macabro que nadie sabe cómo puede terminar.
0.68	Ghost Machine	Un grupo de jóvenes técnicos militares de EE.UU., entran en un complejo donde hay un simulador de combate de alto secreto, con el fin de usarlo para juegos virtuales no autorizados. El sistema se encuentra dentro de una prisión abandonada utilizada para la tortura de prisioneros post-9/11. Pero, pronto, descubren que alguien o algo así ha subido a su propia AI de software. Un jugador nuevo con intereses mortales se ha unido ahora al juego. ¿Cómo sobrevivir o escapar a la masacre incontrolable de un enemigo que es imparable? La batalla final comienza dentro de la máquina GHOST.

Figura 80. Argumentos similares obtenidos con un texto proporcionado por parámetro.

Texto a explorar:

Año 2084. Después de décadas de sangrientas luchas, una fuerza invasora alienígena está a punto de destruir lo poco que queda del mundo. En un desesperado intento por salvar la raza humana, un soldado de la guerrilla, de nombre Miri, recurre a la máquina del tiempo. A sólo 72 horas del plazo límite previsto para el inicio de las Guerras Alienígenas, convence a Miyamoto, un francotirador experto en artes marciales, para que se una a su causa.

similarity	title	overview_es
0.71	Jiu Jitsu	Cada seis años, una antigua orden de luchadores de jiu-jitsu une fuerzas para luchar contra una feroz raza de invasores alienígenas. Pero cuando un célebre héroe de guerra cae derrotado, el destino del planeta y la humanidad está en juego.
0.71	超時空要塞マクロス 愛・おぼえていますか	Una raza alienígena, los Zentraedi, atraviesan la galaxia con destino a la Tierra. Con un ejército de naves que se cuenta por millones y una tecnología mucho más avanzada que la de los humanos, los Zentraedi no necesitarán demasiado tiempo para acabar con la raza humana y recuperar una de sus naves, que se estrelló en el planeta tiempo atrás. Los humanos no tienen mucho donde escoger para escaparse de su aniquilación, así que deciden salvar a los habitantes de la isla donde cayó "Macross", la nave alienígena, usándola para abandonar el planeta, dirigiéndose más allá de los límites del Sistema Solar. Siendo los Zentraedi los nuevos amos del planeta, sólo el tiempo dirá si alguna vez podrán volver a la Tierra y vencerles.
0.70	Pacific Rim: Uprising	Un futuro cercano. Después de la primera invasión que sufrió la humanidad, la lucha aun no ha terminado. El planeta vuelve a ser asediado por los Kaiju, una raza de alienígenas colosales que emergen desde un portal interdimensional, con el objetivo de destruir a la raza humana. Ante esta nueva amenaza para la cual los humanos no están preparados, los Jaegers, robots gigantes de guerra pilotados por dos personas para sobrellevar la inmensa carga neural que conlleva manipularlos, ya no están a la altura de lo que se les viene encima. Será entonces cuando los supervivientes de la primera invasión, además de nuevos personajes como el hijo de Pentecost (John Boyega), tendrán que idear la manera de sorprender al enorme enemigo, apostando por nuevas estrategias defensivas y de ataque. Con la Tierra en ruinas e intentando reconstruirse, esta nueva batalla puede ser decisiva para el futuro.

Figura 81. Argumentos similares obtenidos con un texto proporcionado por parámetro.

4.2. Exploración de artículos de investigación

Este conjunto de datos contiene información sobre artículos de investigación de la página web arXiv¹. Se usará el modelo *all-MiniLM-L6-v2* para calcular los *sentence-embeddings* de los *abstracts* o resúmenes de los artículos y, posteriormente, utilizar la herramienta del SBMC que permite explorar en tiempo real para descubrir artículos similares basándose en el *abstract* y poder evaluar el funcionamiento de la herramienta sobre este conjunto.

En la Figura 82 podemos ver un ejemplo de exploración y se puede observar que los elementos recuperados son bastante buenos. En la Figura 83 y Figura 84 se pueden ver ejemplos de exploración en base al título del artículo, el primero con un texto propio dado por parámetro y el segundo dado otro artículo del conjunto.

Texto a explorar:

Image retrieval using state of the art techniques

similarity	titles	abstracts
0.76	A Comparative Analysis of Retrieval Techniques In Content Based Image Retrieval	Basic group of visual techniques such as color, shape, texture are used in Content Based Image Retrievals (CBIR) to retrieve query image or subregion of image to find similar images in image database. To improve query result, relevance feedback is used many times in CBIR to help user to express their preference and improve query results. In this paper, a new approach for image retrieval is proposed which is based on the features such as Color Histogram, Eigen Values and Match Point. Images from various types of database are first identified by using edge detection techniques. Once the image is identified, then the image is searched in the particular database, then all related images are displayed. This will save the retrieval time. Further to retrieve the precise query image, any of the three techniques are used and comparison is done w.r.t. average retrieval time. Eigen value technique found to be the best as compared with other two techniques
0.76	Comparative Study and Optimization of Feature-Extraction Techniques for Content based Image Retrieval	The aim of a Content-Based Image Retrieval (CBIR) system, also known as Query by Image Content (QBIC), is to help users to retrieve relevant images based on their contents. CBIR technologies provide a method to find images in large databases by using unique descriptors from a trained image. The image descriptors include texture, color, intensity and shape of the object inside an image. Several feature-extraction techniques viz., Average RGB, Color Moments, Co-occurrence, Local Color Histogram, Global Color Histogram and Geometric Moment have been critically compared in this paper. However, individually these techniques result in poor performance. So, combinations of these techniques have also been evaluated and results for the most efficient combination of techniques have been presented and optimized for each class of image query. We also propose an improvement in image retrieval performance by introducing the idea of Query modification through image cropping. It enables the user to identify a region of interest and modify the initial query to refine and personalize the image retrieval results.
0.73	Content Based Image Indexing and Retrieval	In this paper, we present the efficient content based image retrieval systems which employ the color, texture and shape information of images to facilitate the retrieval process. For efficient feature extraction, we extract the color, texture and shape feature of images automatically using edge detection which is widely used in signal processing and image compression. For facilitated the speedy retrieval we are implements the antipole-tree algorithm for indexing the images.

Figura 82. Abstracts de artículos similares obtenidos con un texto proporcionado por parámetro.

¹ <https://arxiv.org/>

Texto a explorar:

Image retrieval using state of the art techniques

similarity	titles
0.76	A Comparative Analysis of Retrieval Techniques In Content Based Image Retrieval
0.76	Comparative Study and Optimization of Feature-Extraction Techniques for Content based Image Retrieval
0.73	Content Based Image Indexing and Retrieval
0.69	Content-based image retrieval using Mix histogram
0.69	Image retrieval approach based on local texture information derived from predefined patterns and spatial domain information
0.69	Texture and Color-based Image Retrieval Using the Local Extrema Features and Riemannian Distance
0.67	Content-Based Image Retrieval Using Multiresolution Analysis Of Shape-Based Classified Images
0.67	Content Based Image Retrieval System using Feature Classification with Modified KNN Algorithm
0.66	Investigating the Vision Transformer Model for Image Retrieval Tasks
0.66	An Efficient Image Retrieval Based on Fusion of Low-Level Visual Features
0.66	Structured Query-Based Image Retrieval Using Scene Graphs
0.65	A Benchmark on Tricks for Large-scale Image Retrieval
0.64	Training Vision Transformers for Image Retrieval
0.64	Dialog-based Interactive Image Retrieval
0.63	Packing and Padding: Coupled Multi-index for Accurate Image Retrieval

Figura 83. Títulos de artículos similares obtenidos con un texto proporcionado por parámetro.

Texto a explorar:

Design of Artificial Intelligence Agents for Games using Deep Reinforcement Learning

similarity	titles
0.73	Deep Reinforcement Learning for General Video Game AI
0.65	Deep Reinforcement Learning
0.65	Auto-Agent-Distiller: Towards Efficient Deep Reinforcement Learning Agents via Neural Architecture Search
0.64	Playing Atari with Deep Reinforcement Learning
0.63	Multi-Agent Deep Reinforcement Learning with Human Strategies
0.62	Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models
0.62	Deep Reinforcement Learning for Adaptive Learning Systems
0.62	Optimizing the Neural Architecture of Reinforcement Learning Agents
0.62	Multi-Agent Deep Reinforcement Learning with Adaptive Policies
0.62	Imagination-Augmented Agents for Deep Reinforcement Learning
0.61	A Deep Reinforcement Learning Architecture for Multi-stage Optimal Control
0.61	A Survey on Deep Reinforcement Learning for Data Processing and Analytics
0.61	Using a Deep Reinforcement Learning Agent for Traffic Signal Control
0.61	Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications
0.61	Deep Reinforcement Learning: An Overview

Figura 84. Títulos de artículos similares a otro artículo del conjunto.

4.3. Exploración de software en base a su descripción

Este conjunto de datos está compuesto por descripciones de aplicaciones móviles. Se utilizará la herramienta del SBMC que permite explorar en tiempo real para poder evaluar el funcionamiento de la herramienta sobre este conjunto a la hora de realizar una búsqueda de una aplicación describiéndola en unas pocas palabras.

En la Figura 85 y Figura 86 podemos ver ejemplos de esta exploración, pero esta vez utilizando un modelo multilenguaje (*paraphrase-multilingual-MiniLM-L12-v2*) para calcular los *sentence-embeddings*. De esta forma las consultas pueden estar en español, aunque los elementos del conjunto estén en otros idiomas.

Observando los resultados de la exploración, con la ayuda del SBMC podría crearse un sistema de recomendación o de búsqueda de aplicaciones sin necesidad de saber el título de esta, bastando con describir lo que hace o la funcionalidad que se busca.

Texto a explorar:

aplicación para cortar un vídeo en trozos y ponerle música de fondo

similarity	description
0.88	Video Trimmer it's very easy App to Cut movie / Add audio to video and change music of any Video files, just select video and audio and add song to video in always possible. This video cutter app cuts & trims selected start time and end time of the video with multiple parts. So you can easily remove the unnecessary parts of video clip. The easiest video cutter app can cut video file and remove unwanted parts like commercials, outtakes and trailers. Effortlessly import and edit videos and export it, you can also Change audio from video and add music file to clip and combine videos . Features : - You can cut & trim Video - Mix audio and video - Add Song to video clip - Speed video and slow motion - Replace Audio of Video - Audio Video Mixer - MP4 Cutter - Mute Video - Add Audio to Video - Merge videos easily - Share your clips to anywhere Download this app and create awesome video to share with your friend and update you video profile Thank you.
0.84	This is a free MP4 Video Cutter and audio cutter application that allows you to cut videos and MP3 files easily. Just choose the starting and ending point of the video and cut it. In the same way you can cut the MP3 file and set it as a ringtone. It can be used for ringtone maker as well. Super smart video editor is here now. Download and start trimming videos now.
0.84	With this application you can easily cut your audio and video files. You can set the file length. Audio trimmer Video Trimmer Audio cutter Video cutter Cut an audio file Cut a video file Trim an audio file Trim a video file
0.83	Video Cutter is an simple and sober app for video trimmer purpose. You can also extract audio from video with this app. You can make video to audio with this video cutter app. You can easily and quickly trim your videos. In few steps you can trim your video. You can share the trimmed video file from the app easily. Step 1) Select the video Step 2) Select the part you want to trim or cut. Step 3) Preview the video (Optional) Step 4) Click on save button. ***** You can also convert video to audio quickly with our this video to audio converter app Step 1) Select the video Step 2) Select the part you want to save as audio. Step 3) Preview the audio (Optional) Step 4) Click on save button. * We hope you like our Video Cutter app and we want to make it better for you. In order to do that and to support the developer, this application has ad in it. Permissions "INTERNET" and "NETWORK ACCESS" are required only to display advertisement.
0.83	This video cutter application cuts & trims selected start time and end time of the video with multiple parts. So you can easily remove the unnecessary parts of video clip. The easiest video cutter app can cut video file and remove unwanted parts like commercials, outtakes and trailers. Effortlessly import and edit videos and export it, you can also Change audio from video and add music file to clip. Features: - Cut video with selected time intervals with same video. - Video Cutter app is very fast and easy to use with user friendly UI. - Mix Audio With Video . - Supports many video formats to trim your video - Save your video to sdcard folder in your phone. - Share video clips. (Send e-mail, upload to YouTube, facebook, twitter, instagram etc.) - Play and Preview video clips - Rename video clips - Delete video clips

Figura 85. Descripciones de aplicaciones similares obtenidas con un texto proporcionado por parámetro.

Texto a explorar:

aplicación para ver las estrellas

similarity	description
0.75	Point your phone or tablet like a magic lens into the night sky, and see in real time what stars and constellations hover above. Starlight is a streamlined sky viewer that makes stargazing exciting and educational. See planets, stars, and constellations come to life against a vibrant backdrop of stars. This is your map to the cosmos! Amaze your friends and educate your kids with this useful app. FEATURES: ...
0.74	The Night Sky View is a magical app that enables you to identify the stars, planets, galaxies, constellations, and even satellites you can see above. All you have to do is point out the star at the night sky and Star Chart will tell you exactly what you are looking at. Even if your view is obscured by clouds or daylight, The Sky Star Finder app will know which stars,...
0.73	Hey, get outdoor with your friends and start star gazing! Let StarTracker guide you to explore the universe. Just hold up and point the device to the sky and have fun! You can see any stars, constellations and deep sky objects you are watching in reality. ===== What our user said: "Best star gazing app I've ever used. Very well put together and great visuals!!! From end to end this..."
0.73	The application allows you to recognize the stars in the sky. Often the brightest stars are visible in the evening, but it is difficult to guess to what constellation they belong - Lyra, Cygnus, or Eagle. Or maybe it's Venus, Mars or Jupiter? This question can answer this application. 3D map * Indicates Zenith Nadir, the compass, the constellations, stars and horizontal grid coordinates. * planetarium. Moon * model of...
0.73	Formerly "Stars and Universe" Start the app and point your device in the direction of the night sky, and the name of the star will be shown. - Features 88 constellations All solar system planets Hipparcos Catalogue nearly 120,000 stars Over 100 Messier objects 100 Brightest Satellites Starlink satellites Comets Dwarf planets and Asteroids Johannes Hevelius constellation images Meteor Shower Satellite notifications - The augmented reality view works even if...

Figura 86. Descripciones de aplicaciones similares obtenidas con un texto proporcionado por parámetro.

Texto a explorar:

Are you a professional of wood and especially of carpentry, do you love building and DIY, your hobby is making wooden furniture? Carpentry is one of the oldest trades in the world. For hundreds of years, people have used wooden logs and turned them into impressive structures and functional, beautiful furniture. Follow the tutorials that we included in our application to start becoming a professional carpenter or if you prefer, you can use carpentry as a hobby.

similarity	description
0.64	Wood Craft Cut And Paint Right now of fulfilling designs and creative levels, you can challenge yourself, your cerebrum, and your companions! It's you against material science and the precarious levels may want to you consider that wooden carving might be such an interesting sport? do that Wood Craft Cut And Paint game and you may be surprised with the undertaking that supplied. carved timber in any such ideal form...
0.63	Have you ever felt like a Carpenter, making wood stuff by your own ? Let's show us your carpenter skills and making awesome wood stuff. Also becarefull with some tricky customer, they sometime give you a bad deal for your work. Prove your Carpenter skills with hundreds of handmade levels!...
0.61	We would love to continue the series of wood carving games! We've implemented the real physics for woodworking machine to bring you more joy. Wood carving tool is based on a mix of a dremel moto-saw and drill press with beaver (sideways cutting) drill saw bit. This time no carving knives - just wood engraving. At each level you are given a sample to be cut out of the workpiece....
0.61	We would love to continue the series of wood carving games! We've implemented the real physics for woodworking machine to bring you more joy. Wood carving tool is based on a mix of a dremel moto-saw and drill press with beaver (sideways cutting) drill saw bit. This time no carving knives - just wood engraving. At each level you are given a sample to be cut out of the workpiece....
0.61	for wood lovers! It's your satisfying time, our lather simulator will help you to turn, cut and paint the wood. Time to be a real carpenter. Enjoy!...

Figura 87. Descripciones de aplicaciones similares a otra dada.

4.4. Detección de tweets con sentimiento depresivo

Este conjunto de datos consiste en aproximadamente 200.000 tweets etiquetados con sentimiento depresivo o no depresivo.

El objetivo es poder detectar tweets que muestren un carácter depresivo utilizando las funcionalidades que ofrece el SBMC para explorar las similitudes entre dos conjuntos de datos. Para ello, los tweets del conjunto original se dividen en dos conjuntos independientes:

- **Depression_tweets.csv**: contiene 108.040 textos de tweets con sentimiento depresivo.
- **Tweets_to_explore.csv**: contiene 16.037 textos de tweets, de los cuales 7.797 están etiquetados como depresivos y el resto como no depresivos.

Una vez los dos conjuntos están preparados se usa el modelo *all-MiniLM-L6-v2* para realizar los *sentence-embeddings* del conjunto de tweets con sentimiento depresivo, formando una base de datos con los que comparar el texto que se desea clasificar como se ilustra en la Figura 88.

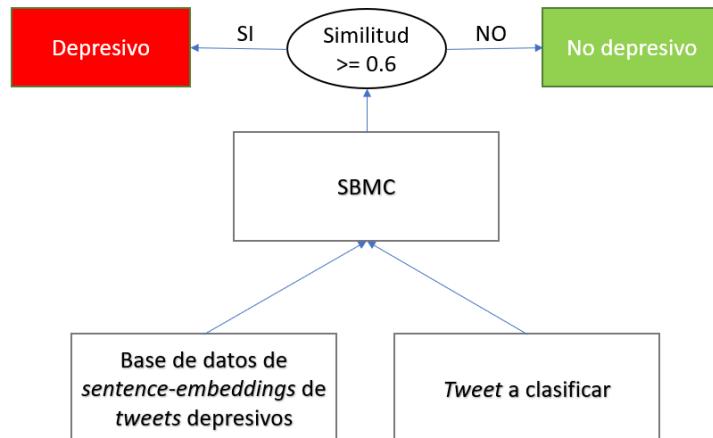


Figura 88. Arquitectura del clasificador de tweets depresivos.

Para estudiar este posible enfoque de clasificación se utiliza la funcionalidad del SBMC que permite calcular las similitudes entre dos conjuntos, obteniendo así la matriz de similitud entre el conjunto de tweets a explorar y el conjunto de tweets depresivos.

Una vez se obtiene la matriz de similitud, los tweets a explorar se clasificarán como depresivos o no en base a lo que se parecen a las del conjunto que solo contiene

depresivos. Así, se establece un valor de corte para considerar que las que se parezcan al menos un 60% a alguna del conjunto serán depresivas.

Una vez clasificados mediante el anterior criterio los tweets del conjunto de prueba obtendremos la matriz de confusión que se puede observar en la Figura 89, donde la clase depresiva está representada por el número 0 y la restante por el 1. Esta matriz cuenta con cuatro posiciones al ser una clasificación binaria, en la que la clase positiva se refiere a tweets con sentimiento depresivo y la clase negativa a los restantes:

- **Positivos Verdaderos:** es el número de elementos que se han predicho como clase positiva y realmente eran de esta clase.
- **Negativos Verdaderos:** es el número de elementos que se han predicho como negativos y realmente eran de la clase negativos.
- **Positivos Falsos:** es el número de elementos de la clase negativa que se han predicho como positivos.
- **Negativos Falsos:** es el número de elementos de la clase positiva que se han predicho como negativos.

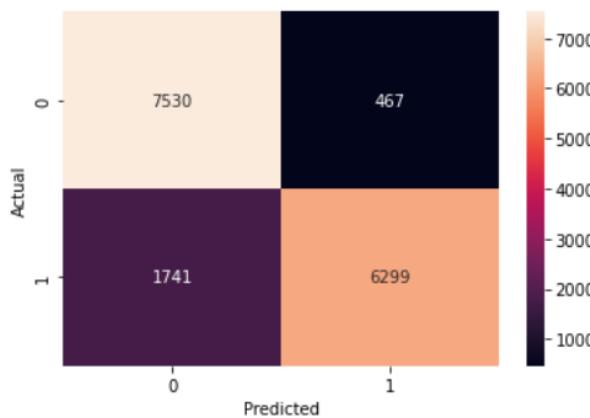


Figura 89. Matriz de confusión resultante.

Para evaluar los resultados obtenidos, se calculan las métricas *precisión*, *recall* y *F-Measure* [31] para cada clase:

- **Precision o precisión:** es la habilidad del clasificador para no etiquetar un elemento de la clase negativa como de la clase positiva. Se calcula como $\frac{pv}{pv+fp}$, siendo vp el número de Positivos Verdaderos y fp el número de Negativos Falsos.

- **Recall o exhaustividad:** es la habilidad del clasificador para encontrar todos los elementos de la clase positiva. Es decir, el número de elementos que se han clasificado bien del total de positivas. Se calcula como $\frac{pv}{pv+nf}$, siendo vp el número de Positivos Verdaderos y fn el número de Negativos Falsos.
- **F-Measure o Valor-F:** es una medida que combina las dos métricas, exhaustividad y precisión, proporcionando un valor entre 0 y 1, siendo 1 el máximo valor posible y 0 el peor. Se calcula como $\frac{2*exhaustividad*precisión}{exhaustividad+precisión}$.

Los valores de las métricas se pueden observar en la Figura 90 y se han realizado las diferentes observaciones al respecto:

- La métrica exhaustividad indica que 94,2% de los tweets con sentimiento depresivo y el 78,3% de los tweets sin sentimiento depresivo se clasifican correctamente. En este caso se ha de considerar que es más importante detectar un texto depresivo correctamente que uno que no lo es, por lo que la exhaustividad del modelo es bastante buena.
- La métrica precisión indica que el 81,2% de los tweets predichos como depresivos eran realmente de esta clase y el 93,1% de los tweets predichos como no depresivos eran realmente no depresivos. Se considera que es importante no equivocarse al clasificar uno como no depresivo mientras si lo era, por lo que una precisión del 93,1% en este caso es bastante buena.
- La métrica *accuracy* o exactitud indica que el 86,2% del total de elementos son bien clasificados. Esta se calcula como número de elementos bien clasificados entre el número total de elementos.
- La métrica *F-Measure* de ambas clases tiene un valor bastante alto indicando un 87,2% para el conjunto de depresivos y un 85,1% para el otro conjunto.

	precision	recall	f1-score	support
Depressive	0.812	0.942	0.872	7997
Non-Depressive	0.931	0.783	0.851	8040
accuracy			0.862	16037
macro avg	0.872	0.863	0.862	16037
weighted avg	0.872	0.862	0.861	16037

Figura 90. Métricas obtenidas a partir de la matriz de confusión.

Observando los buenos resultados llegamos a la conclusión de que cuantos más *sentence-embeddings* de textos con sentimiento depresivo haya en la base de datos mejor detectará otros textos con este sentimiento.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

A la vista de los resultados obtenidos en el SBMC, podemos tener en cuenta varias características. Primero, llegamos a la conclusión de que el modelo seleccionado influye mucho en los resultados obtenidos. La elección de un modelo u otro puede hacer que las soluciones arrojadas sean totalmente diferentes, pero ambas válidas. Además, tras realizar pruebas con modelos multilenguaje, comprobamos que estos funcionan muy bien encontrando similitudes entre textos de distintos idiomas, por lo que se recomienda su uso cuando nuestro conjunto de datos no tenga un idioma claramente definido. Asimismo, hemos concluido que el SBMC devuelve resultados coherentes independientemente del conjunto de datos que se le aporte sin olvidar la importancia que tiene elegir el modelo correcto cuando el lenguaje es muy específico (por ejemplo, si tratamos con textos médicos será mejor utilizar un modelo entrenado en diferentes datos médicos como Med-BERT [32] que aquel que haya sido entrenado en comentarios de la red social Reddit).

Por otro lado, en cuanto al entrenamiento de los modelos hemos llegado a la conclusión de que este es prácticamente inútil cuando el conjunto de datos con el que se le va a poner a prueba contiene un lenguaje cotidiano. Esto se debe a que el modelo ya ha sido entrenado con billones de sentencias y conoce el lenguaje en general, por lo que entrenarlo sobre un par de sentencias más no tiene un gran efecto en el modelo. En cambio, cuando el lenguaje de nuestro *dataset* es más específico como por ejemplo documentos biológicos o queremos realizar una tarea en específico con él como pueden ser responder preguntas o completar texto, creemos que un entrenamiento podría hacer que nuestro modelo se comportara mejor ante este lenguaje o tarea en particular (aunque, en este trabajo, no hemos realizado experiencias al respecto).

Tras realizar distintas pruebas con nuestra herramienta, descubrimos la cantidad de utilidades prácticas que tiene. Como hemos visto anteriormente, podemos aplicarlo a películas para navegar entre similares, a tweets para detectar sentimientos depresivos o a distintos informes médicos para diagnosticar enfermedades rápidamente. Con imaginación, las aplicaciones prácticas de la herramienta desarrollada son muy amplias.

Por otra parte, en cuanto a la eficiencia de los *sentence-transformers*, cabe destacar que arrojan muy buenos resultados para prácticamente cualquier texto que se le enfrente y de forma muy rápida, sobre todo cuando estos son calculados en un hardware potente o dedicado específicamente a la Inteligencia Artificial. Además, comprobamos que la paralelización del trabajo ayuda considerablemente a reducir los tiempos de ejecución. Sin embargo, no siempre es posible pues nos hemos

encontrado muchas veces con serias limitaciones de RAM y VRAM de modo que muchas tareas se han tenido que ejecutar por lotes.

En conclusión, estos modelos disponibles que ya están entrenados ahorran mucho tiempo al ser ya capaces de entender el lenguaje y únicamente tener que adaptarlos a una tarea o dominio y no enseñarle todo desde cero. Disponer de ellos agiliza el trabajo de muchos proyectos y ahorra un gasto de tiempo de cómputo y energía enorme. Además, estas técnicas de Procesamiento del Lenguaje Natural se encuentran actualmente en auge y prácticamente cada día surgen nuevas herramientas como GPT-3 que se instalan en nuestro día a día sin darnos a penas cuenta.

5.2. Trabajo futuro

El trabajo futuro se divide en dos líneas principales. La primera se basa en las mejoras que podríamos realizar al SBMC y la segunda son las que podríamos implementar para que nuestro explorador de películas basado en similitud sea una mejor aplicación, más completa, rápida y exacta.

5.2.1. Trabajo futuro en el SBMC

En cuanto al trabajo futuro que podría realizarse en el SBMC, uno de los aspectos más importantes es la creación de una interfaz web y un servidor independiente del conjunto de datos aportado. Esta interfaz web permitiría a un usuario cargar su *dataset*, sea cual sea su contenido y, tras esperar la ejecución del algoritmo y la exportación de sus resultados a la base de datos, el usuario podría navegar por una interfaz mucho más cómoda y menos primitiva que la actual. Cabe destacar que, con conjuntos de datos muy grandes, el tiempo de ejecución del algoritmo y su exportación a la base de datos con un fichero como el de las películas con 584.500 entradas puede llegar a tardar hasta unos 30 minutos, por lo que el usuario debería esperar un tiempo considerable desde que suba el fichero hasta poder explorarlo.

Por otro lado, el SBMC permite actualmente una exploración primitiva e independiente del *dataset* aportado, ya sean datos médicos, resúmenes de libros, actas deportivas, etc. Por ello, para asegurarnos de que el modelo seleccionado en nuestra herramienta es capaz de reconocer y entender el lenguaje específico del *dataset*, se podría realizar un entrenamiento previo a la generación de *sentence-embeddings* utilizando las técnicas de Augmented SBERT en las que no necesitamos más que un *dataset* sin datos etiquetados, en este caso el aportado por el usuario.

Además, como objetivo futuro, se podría ampliar la herramienta permitiendo comparar no solo las descripciones textuales de los contenidos multimedia, si no otro tipo de datos como pueden ser imágenes, vídeos y audios.

5.2.2. Trabajo futuro en el explorador de películas

Actualmente, nuestro explorador de películas es una aplicación que permite la búsqueda de películas según su sinopsis, título y actores. Además, únicamente permite realizar esta búsqueda en un único idioma, el inglés. Por tanto, gran parte de las mejoras que podríamos realizar se basarían en mejorar esta búsqueda, añadiendo más filtros como el *rating* de las películas, fecha de estreno, director o duración, permitiendo una búsqueda más exacta a lo que quiere el usuario y mejorando así su experiencia. También se podrían crear nuevas vistas en la web como la opción de ver toda la información de un actor junto con sus películas u otros actores relacionados.

A su vez otra de las grandes implementaciones sería permitir realizar esta búsqueda en distintos idiomas, principalmente el castellano ya que es el más usado localmente. Para ello deberíamos explorar diferentes modelos multilenguaje para que los *sentence-embeddings* de un mismo argumento en distintos idiomas sean reconocidos como similares.

Por último, en cuanto a la optimización en tiempo y espacio del servidor creemos que las consultas de la base de datos y la ejecución del algoritmo son suficientemente rápidas. Sin embargo, nos encontramos con limitaciones en el espacio de la memoria de vídeo donde se almacenan los *sentence-embeddings* para la ejecución del algoritmo en tiempo real. Una memoria de vídeo de 6GB no es suficiente para tener almacenados todos los *sentence-embeddings* y ejecutar el algoritmo por lo que habría que investigar las posibilidades de carga y descarga de estos datos según las peticiones recibidas u otras que disminuyan el consumo de memoria VRAM.

5. Conclusions and future work

5.1. Conclusions

In view of the results obtained in the SBMC, we can take into account several characteristics. First, we conclude that the selected model greatly influences the results obtained, the choice of one model or another can make the solutions thrown out totally different, but both valid. In addition, after testing with multilanguage models, we found that they work very well by finding similarities between texts from different languages, so it is recommended to use them when our data set does not have a clearly defined language. Likewise, we have concluded that the SBMC returns consistent results regardless of the data set that is provided without forgetting the importance of choosing the right model when the language is very specific. For example, if we deal with medical texts it will be better to use a model trained in different medical data such as Med-BERT [32] than one that has been trained in comments from the social network Reddit.

On the other hand, in terms of training or *fine-tuning* we have come to the conclusion that this is practically useless when the data set with which it is going to be tested contains an everyday language. This is because the model has already been trained with trillions of sentences and knows the language in general, so training it on a couple more sentences does not have a great effect on the model. On the other hand, when the language of our *dataset* is more specific such as biological documents or we want to perform a specific task with it such as answering questions or completing text, we think that a *fine-tuning* can make our model behave better before this particular language or task (nevertheless, in this work we have not conducted any experiment in this sense).

After performing different tests with our tool, we discover the amount of practical utilities it has. As we have seen before, we can apply it to movies to navigate between similar, to tweets to detect depressive feelings or to different medical reports to diagnose diseases quickly. With imagination, the practical applications of the developed tool are practically endless.

On the other hand, in terms of the efficiency of *sentence-transformers*, it should be noted that they yield very good results for practically any text that faces it and very quickly, especially when they are calculated on a powerful hardware or dedicated specifically to Artificial Intelligence. In addition, we found that the parallelization of the work helps considerably to reduce the execution times. However, it is not always possible because we have often encountered serious limitations of RAM and VRAM so that many tasks have had to be executed in batches.

In conclusion, these available models that are already trained save a lot of time by already being able to understand the language and only having to adapt them to a

task or domain and not teach them everything from scratch. As a consequence it streamlines the work of many projects and saves a huge cost of computing time and energy. In addition, these Natural Language Processing techniques are currently booming and practically every day new tools such as GPT-3 emerge that are installed in our day to day without realizing it.

5.2. Future work

Future work is divided into two main lines. The first is based on the improvements we could make to the SBMC. The second is what we could implement to make our similarity-based movie browser a better, more complete, faster, and more accurate application.

- **Future work in SBMC**

As for future work that could be done on the SBMC, one of the most important aspects is the creation of a web interface and server independent of the dataset provided. This web interface would allow the user to upload his dataset, whatever its content, and, after waiting for the execution of the algorithm and the export of its results to the database the user could navigate through a much more comfortable and less primitive interface than the current one. It should be noted that this idea is unfeasible to implement with very large datasets, since the time to run the algorithm and export it to the database with a file such as the movie with 584,500 entries, can take up to 30 minutes.

On the other hand, SBMC currently allows for primitive and independent exploration of the provided dataset, be it medical data, book summaries, sports records, etc. Therefore, to ensure that the model selected in our tool is able to recognize and understand the specific language of the dataset, a small fine-tuning could be performed prior to the generation of *sentence-embeddings* using Augmented SBERT techniques in which we only need a dataset without labeled data, in this case the one provided by the user.

In addition, as a future goal, the tool could be extended to allow comparison not only of textual descriptions of multimedia content, but also of other types of data such as images, videos, and audios.

- **Future work in movie explorer**

Currently, our movie browser is an application that makes it possible to search for movies according to their synopsis, title, and actors. Moreover, it only allows this search in one language, English. So, most of the improvements we could make are based on improving this search, adding more filters such as movie rating, release date, director, or duration, allowing a more tailored search to what the user wants and improving their experience. New web views could also be created such as the option to see all the information of an actor along with their movies or other related actors.

Another great implementation would be to allow this search in different languages, mainly Spanish as it is one of the most important. For this we should explore different multilanguage models so that *sentence-embeddings* of the same plot in different languages are recognized as similar.

Finally, regarding time and server space optimization, we believe that database queries and algorithm execution are fast enough. However, we found limitations in the video memory space where the *sentence-embeddings* are stored for real-time execution of the algorithm. A video memory of 6GB is not enough to store all the *sentence-embeddings* and execute the algorithm, so it would be necessary to investigate the possibilities of uploading and downloading this data depending on the requests received or others that reduce the VRAM memory consumption.

Contribuciones individuales al proyecto

Dados los requerimientos mínimos del proyecto, un sistema de exploración de contenidos multimedia que permita a los usuarios navegar entre las distintas entradas, dividimos el trabajo a realizar en dos subproyectos:

- La parte encargada de reunir los contenidos multimedia y ordenarlos según su similitud. Lo que llamaríamos *back-end*, al ser la parte del proyecto no visible para el usuario final.
- La parte encargada de recoger la información reunida en el *back-end* y disponerla al usuario en un formato adecuado y fácil de usar. Lo que llamaríamos *front-end*, al ser la parte del proyecto visible para el usuario final.

Según esta estructura decidimos que la mejor manera de trabajar sería la realización de ambos subproyectos en paralelo, pero siendo conscientes de las dependencias que se generarían conforme avanzásemos en nuestro trabajo, como la necesidad de un conjunto de datos inicial para la visualización del primer prototipo de la interfaz, o una propia interfaz inicial para la comprobación del correcto funcionamiento del algoritmo. Una vez acordado el plan de trabajo, y dado el número de integrantes del equipo, decidimos dividirnos en dos grupos: inicialmente un grupo de dos personas, Borja y Óscar, encargado de la parte del *back-end*, y un segundo grupo de tres personas, Diego, Jheison y Rafael, para el *front-end*. Aunque cabe destacar en fases más avanzadas del proyecto, la participación de Diego tanto en el grupo encargado del *back-end* en la parte del servidor, como en el de *front-end*.

Rafael Noblejas Pérez

Como ya se ha mencionado anteriormente, mi contribución al proyecto ha sido mayoritariamente al grupo encargado del *front-end*. En fases iniciales del proyecto decidimos explorar dos opciones para la visualización de los datos generados por el algoritmo:

- Una implementación de la interfaz mediante la biblioteca React de JavaScript, como sugerencia por parte del director del proyecto José Luis.
- Una implementación de la interfaz mediante el entorno de desarrollo de Spring, idea sugerida por Diego.

Al ser tres integrantes en el subproyecto del *front-end*, y dada la sugerencia de Diego, decidimos encargarnos Jheison y yo de la implementación en React, y Diego de la implementación en Spring.

Lo primero que decidimos hacer para el desarrollo de la interfaz fue diseñar una serie de bocetos de la versión inicial para establecer los componentes mínimos que queríamos que tuviese la aplicación. Una vez creados, compartimos los mismos con el resto del equipo para recibir todo el *feedback* posible antes de empezar con el desarrollo. Estos bocetos se crearon utilizando la herramienta de Balsamiq Wireframe debido a nuestra familiaridad con ella. Una vez establecido el diseño base de la interfaz pasamos a estudiar la biblioteca de React para empezar el desarrollo, sin embargo, dado nuestro poco conocimiento de la biblioteca de React, realizamos varios tutoriales y ejercicios reunidos en su página web para familiarizarnos con ella. Una vez aprendidos los conceptos básicos necesarios, nos sentimos con la confianza suficiente para empezar a programar la interfaz. Antes de ello, reconocimos la necesidad de reunir todos nuestros avances en la tarea en un mismo repositorio con el fin de facilitar la organización del trabajo, y de mantener un histórico con todos los cambios que se realizarían. Para ello utilizamos GitHub por nuestra familiaridad con la herramienta y por las facilidades que proporciona a la hora de utilizarla en combinación con el editor de código fuente Visual Studio Code¹, utilizado para la programación del código de la interfaz.

Primero decidimos hacer conjuntamente la estructura base de la web y dos pantallas (`<Homepage>` y `<Movie>`) para tener una versión inicial de lo que sería la web. Tras definir los componentes que tendrían estas dos pantallas, decidimos dividirnos la programación de dichos componentes entre los dos. Tras esto, y tras obtener un conjunto de datos inicial para probar el funcionamiento de las dos pantallas, decidimos empezar a desarrollar el resto de las pantallas necesarias y sus componentes, de nuevo dividiendo el trabajo.

Esta versión inicial permitía solo la búsqueda de películas por su título. Las siguientes funcionalidades en implementarse fueron las búsquedas por texto

¹ <https://code.visualstudio.com/>

empleando los dos modelos existentes, idea sugerida por el equipo del *back-end*, como forma de demostrar la capacidad del algoritmo de trabajar también con textos más complejos. Tras ello, decidimos añadir más funcionalidades que complementasen a la aplicación, como la capacidad de añadir filtros de género, y de buscar películas por los actores que aparecen en ellas. De nuevo, estas tareas realizadas de forma conjunta entre Jheison y yo. Paralelamente al desarrollo de la lógica, cada uno fue responsable de programar las hojas de CSS necesarias para cada pantalla/componente, siguiendo una estructura, formas y colores comunes entre ellos.

En cuanto a mi contribución en la memoria del proyecto, me encargué de la redacción del resumen, de parte de la introducción, el plan de trabajo y la estructura del documento, así como de la interfaz de usuario y su implementación. Además de dichos apartados, también me encargué de las traducciones necesarias, formateo de la memoria, y gran parte de su revisión.

Jheison Orellana Díaz

Me he encargado junto a Rafael Noblejas Pérez de la parte *front-end*.

Hemos diseñado en primera instancia unos bocetos, con una aplicación, para determinar qué elementos principales necesita la interfaz. Dicha aplicación utilizada es la misma que utilizamos en la asignatura Desarrollo de Sistemas Interactivos.

Tuvimos que documentarnos primero para el desarrollo de la página, ya que ninguno sabía sobre el entorno. Luego nos decidimos, en conjunto, por una biblioteca de JavaScript, llamada React, debido a su sencillez para desarrollar la página web.

Había que desarrollar dos páginas principales, las cuales nos hemos ido repartiendo el trabajo por igual. Ambos hemos ido desarrollando el código y depurándolo. Para un control de versiones empleamos GitHub, donde guardamos todo el código de la página web y sus versiones.

En un principio, cuando teníamos el primer *dataset* tuvimos que cargar todos los datos antes de que renderice la interfaz. Para ello, hemos tenido que *parsear* de *csv* a una lista. Y con eso conseguimos una primera vista de cómo sería la interfaz web con sus dos páginas principales.

Luego, cuando nuestros compañeros crearon el servidor, diseñamos la página de forma que, a través de consultas, obtenemos los datos de las películas. Dependiendo del desarrollo del servidor, hemos metido nuevas funcionalidades, como otro tipo de búsqueda por texto, título, actores y géneros.

Hemos ido tocando con CSS el diseño de la web, ya sea la forma, los colores, la estructura. Se ha creado un fichero CSS por cada componente para una mejor organización.

Para su desarrollo tuvimos que trabajar junto al equipo de *back-end* debido a que necesitábamos un conjunto de datos iniciales y poder continuar con el desarrollo web.

Después en conjunto, ambos equipos hablamos para la integración de nuevas funcionalidades, como búsqueda por géneros, tipo de búsqueda y método de búsqueda.

Borja Aday Guadalupe Luis

Mis contribuciones individuales al proyecto se centran sobre todo en la parte del back-end, sobre todo en el algoritmo, pero participando también en la base de datos y el servidor.

Cabe destacar que prácticamente todo el back-end lo he realizado con la ayuda de mi compañero Óscar, por lo que hablaré en plural haciendo referencia a ambos.

Inicialmente, nos dedicamos al estudio de BERT y SBERT en la documentación de la web oficial, distintos papers como puede ser el de “Attention Is All You Need.” y diferentes vídeos explicativos en YouTube.

Tras haber estudiado estos documentos y comprender su funcionamiento necesitábamos de un primer dataset con el que empezar a realizar pruebas. Inicialmente creamos un fichero en *Python* encargado de recolectar las películas de IMDB, pero esta idea fue rechazada rápidamente por la falta de una API pública y fue sustituida por TMDb. Este recolector de datos comenzó siendo lineal con un bucle que preguntaba por cada id disponible, posteriormente se mejoró con el uso de hebras cada una de ellas encargada de unos ids concretos y finalmente se utilizó multiprocessing para acelerar aún más el tiempo de ejecución.

Una vez teníamos nuestro primer *dataset* pudimos observar que teníamos grandes inconvenientes en los datos como podían ser argumentos vacíos o inútiles que nos iban a estorbar a la hora de medir similitudes. De ahí surge la idea de crear un archivo que limpiara el *dataset* de datos inútiles. Inicialmente este notebook era muy sencillo y únicamente eliminaba aquellas películas cuyos argumentos fueran vacíos o no incluyeran información relevante. Más adelante, tras haber finalizado nuestra herramienta SBMC, comprendimos que era importante eliminar aquellas películas cuyo argumento no estuviera en inglés, pues el modelo que usamos está entrenado sobre todo en este idioma para lo cual usamos la biblioteca *lang-detect*. Por otro lado, hicimos uso de nuestra propia herramienta para encontrar y borrar argumentos no válidos similares a los conocidos, quedándonos con un *dataset* final con 584.500 películas.

Posteriormente, y ya con datos útiles que pasarle al algoritmo, nos centramos en la implementación de nuestra herramienta, la cual estuvo inicialmente centrada en medir la similitud del *dataset* de películas. En un principio, este notebook constaba únicamente de pocas celdas de código encargadas de calcular los *sentence-embeddings*, calcular las similitudes del coseno de forma rudimentaria y lineal y un *re-ranking* muy simple. Una vez teníamos algo que funcionaba, nos dedicamos a mejorar los tiempos de ejecución. Esto se logró primeramente ejecutando el código generador de *sentence-embeddings* en GPU y sustituyendo el cálculo de similitudes rudimentario por *Faiss*. Finalmente, este archivo se generalizó para ser utilizado con cualquier *dataset* y se añadieron opciones de exportación y guardado de datos y exploración en tiempo real.

Una vez teníamos todos los datos necesarios, nos encontramos con la necesidad de almacenarlos y ponerlos a disposición de la web dejando de depender así de TMDB. Para ello, comenzamos creando las tablas y guardando todos los datos en una base de datos SQL.

Para que estos datos fueran accesibles desde la web, creamos entre Óscar, Diego y yo una primera versión del servidor con la biblioteca *http.server*¹ en *Python* con consultas muy sencillas y con la posibilidad de ejecutar el algoritmo en tiempo real de forma muy rápida, añadiendo muchas funcionalidades a la interfaz web. Fui el encargado de tener esta base de datos y el servidor alojados en mi ordenador de modo que el grupo de desarrollo de la interfaz web pudiera seguir avanzando sin la necesidad de que estuvieran pendientes de ningún cambio en los datos o el código asociado al servidor. La versión final del servidor implementada en *Flask* fue realizada por Diego.

Por último, en cuanto a la memoria, me he encargado junto a Óscar de realizar todo el trasfondo, la arquitectura del proyecto (SBMC y los primeros 6 puntos del explorador de películas), las aplicaciones prácticas y las conclusiones y trabajo futuro (sin incluir las traducciones). Además, entre ambos hemos puesto todas las referencias de la bibliografía y las referencias cruzadas a las figuras.

¹ <https://docs.python.org/3/library/http.server.html>

Óscar Morujo Fernández

Mi contribución en este proyecto se centra en el desarrollo del *backend* junto a mi compañero Borja.

En un principio tuve que investigar las últimas técnicas del Procesamiento del Lenguaje Natural, ya que el objetivo del proyecto es utilizar las técnicas del estado del arte como núcleo. Para ello, hubo una primera fase de investigación en la que me dediqué a entender bien la arquitectura y funcionamiento de los *Transformers*, los modelos que han surgido a partir de su publicación, y lo necesario para poder usarlos.

Una vez avanzada la investigación, hubo una fase en la que se probaron diferentes modelos de HugginFace. Sabiendo que estos modelos usan de GPU para ser más rápidos, investigamos posibles opciones para poder ejecutarlo en cualquier ordenador de manera sencilla, dando finalmente con Google Colaboratory. Investigamos todas las opciones y funcionalidades que ofrecía y decidimos aprender a utilizar la plataforma para desarrollar el explorador en esta.

Junto a Borja, desarrollé el *script* que recoge de internet los datos de las películas que se utilizaran para el explorador web, que en un principio era un *scraper* sobre la web de IMDB, pero finalmente decidimos realizarlo sobre la página de TMDB y utilizar su API disponible en Python.

Además de recopilar el conjunto de datos mencionado, hemos desarrollado un preprocesamiento de este, ya que al ser TMDB una web en la que los usuarios añaden libremente información sobre las películas había muchas sinopsis que no eran útiles para el proyecto. En un principio se eliminaron las que se observaba fácilmente que no eran válidas y que se repetían con más frecuencia. Más adelante con las primeras versiones del SBMC realizamos una limpieza más profunda aplicando la herramienta para encontrar otras sinopsis similares a las que habíamos clasificado como no validas permitiéndonos encontrar otras muchas que no habíamos eliminado en la primera limpieza.

He desarrollado con mi compañero Borja la herramienta SBMC, que en un principio estaba diseñada para funcionar con el conjunto de datos de películas mencionado, pero que tras unas varias versiones conseguimos convertirlo en una herramienta más independiente de los datos. Además, nos hemos encargado de la generación de la base de datos de la web de películas del proyecto con el SBMC.

Tras una primera versión completa del algoritmo, empezamos una fase en la que se intenta mejorar el rendimiento e interfaz de la aplicación. Tras investigar, nos encargamos de modificar ciertas funciones de nuestro algoritmo para introducirle las siguientes mejoras:

- Uso de Faiss para mejorar el tiempo de ejecución de cálculo de similitudes.
- Paralelización de procesos para mejorar el tiempo de ejecución general.

- Uso de Pandas para mejorar el tiempo de subida de datos a una base de datos.

También me he encargado de realizar los entrenamientos de modelos con diferentes técnicas para intentar mejorar los resultados de la exploración de TMDB y de recopilar de los diferentes conjuntos de datos sobre los que aplicar el SBMC para realizar las pruebas reflejadas en el punto de Aplicaciones Prácticas:

- Detección de tweets con sentimiento depresivo.
- Artículos de investigación de arXiv.
- Descripciones de aplicaciones móviles.

Respecto al redactado de la memoria, me he encargado junto a Borja de todo lo relativo al *backend*: el trasfondo del proyecto, la herramienta SBMC, el explorador de películas basado en la similitud y el punto de aplicaciones prácticas; además de colaborar en el redactado de la introducción y resumen del proyecto.

Diego Enrique de Miguel López

Mi trabajo ha consistido principalmente en el desarrollo del servidor y la base de datos realizando también pequeñas modificaciones a la web y la creación del notebook *DataBaseLoaderTMDb*, que permite la correcta subida y configuración del dataset final para el correcto funcionamiento de la aplicación. Cabe destacar la labor de Borja que ha hecho de unión y entre el desarrollo del algoritmo, el servidor y la base de datos; y el trabajo de Jheison ya que juntos hemos podido unificar la web junto con el servidor de forma eficiente y rápida.

Cuando Borja y Óscar obtuvieron su primer *dataset* de similitudes nos vimos con la necesidad de comenzar el desarrollo de la aplicación web para poder comparar los datos de forma eficiente. Para ello, al no tener conocimientos sobre los distintos *frameworks* decidimos implementar en paralelo las 2 opciones básicas que teníamos. Por un lado, Jheison y Rafael comenzaron a programar la web utilizando *React*, idea sugerida por José Luis, el director del proyecto. Por el otro yo comencé el desarrollo en *Angular* y *Spring*, bibliotecas que conocía vagamente gracias a las prácticas de empresa que estaba cursando.

Tras comparar ambos proyectos decidimos quedarnos con la web programada usando *React* y desarrollar un nuevo servidor en *Python* ya que esto nos permitía fusionar el algoritmo que estaban desarrollando Borja y Óscar junto con el nuevo servidor.

En la primera versión del servidor usamos *http.server*, una biblioteca de *Python* muy primitiva cuya escalabilidad era muy baja. Investigando encontré varias opciones y por su simplicidad decidí refactorizar el servidor e implementarlo usando *Flask*. Este proporciona una conexión continua y simple con la base de datos, ya que no tenemos que gestionarla nosotros. También mejora considerablemente la simplicidad de los *endpoint*, ya que las rutas no es necesario *parsearlas*; en cambio con *http.server* debíamos crear expresiones regulares para que funcionaran correctamente.

En paralelo implementé la base de datos, que al principio tenía una única tabla, con todos los datos de las películas. Pero cuando Jheison y Rafael crearon el filtro de géneros tuve que rediseñar la base de datos añadiéndole dos tablas más, la tabla de géneros y una tabla intermedia que asocia las películas a cada uno de los géneros.

Al principio las consultas eran poco eficientes, en el peor de los casos una consulta podía tardar hasta 20 segundos en realizarse, por lo que investigando descubrí junto con Borja los índices *Fulltext* que mejoraban sustancialmente la mayoría de las consultas. Aun así, las consultas que necesitaban filtrar por género eran bastante lentas. La última modificación fue refactorizar estas consultas construyéndolas con los operadores *MATCH* y *AGAINST*, los cuales son únicos para las consultas con campos que contienen índices *Fulltext*.

Por último, decidí crear el notebook *DataBaseLoaderTMDB* a partir de un script que había realizado en *Python* y que permitía cargar toda la base de datos y configurarla, añadiéndole los índices necesarios, para que funcionara sin problemas con cualquier actualización del dataset de películas.

Apéndice

A continuación, se enumeran los pasos necesarios para la generación de la base de datos, el arranque del servidor, y la inicialización de la aplicación web para poner en funcionamiento el explorador de películas basado en la similitud.

Generación de la base de datos

Generar la base de datos es un proceso sencillo que se realiza siguiendo los siguientes pasos:

1. Descomprimir la carpeta “Base de datos”.
2. En el gestor de base de datos se debe crear un *schema* dedicado a este proyecto.
3. Acceder a la carpeta “Base de datos” extraída anteriormente y modificar el archivo constants.py con los parámetros que permitirán la conexión con la base de datos.
4. En una consola del sistema operativo acceder a la carpeta “Base de datos” y ejecutar las siguientes líneas de código:

```
pip install pandas
pip install sqlalchemy
pip install pymysql
python DataBaseLoaderTMDB.py
```

5. Este programa creará una conexión con la base de datos y cargará los siguientes ficheros en ella:
 - *genres_ids.csv*: relaciona cada género con un id.
 - *movies_genres.csv*: indica para cada película a qué géneros pertenece.
 - *dataset.csv*: contiene la información de las películas.
 - *similarity_matrix*: contiene la matriz de similitud entre películas.
6. Tras unos minutos se dispondrá de toda la información y las optimizaciones necesarias en la base de datos para el correcto funcionamiento del servidor.

Inicialización del servidor

Ejecutar el servidor es uno de los pasos más complicados. Este se puede ejecutar sin necesidad de tener una GPU, pero es recomendable configurarla para que todo corra lo más fluido posible.

Para instalar la GPU se deben seguir las siguientes instrucciones:

1. Descargar e instalar los CUDA Drivers de Nvidia disponibles en la web: <https://developer.nvidia.com/cuda-downloads>.
2. En la web oficial de PyTorch: <https://pytorch.org/> se debe seleccionar los parámetros correctos para instalar esta biblioteca. Es importante seleccionar CUDA como plataforma de cómputo para instalar la versión GPU. Una vez seleccionados todos los parámetros se copia la instrucción de instalación y se ejecuta en una consola del sistema operativo.
3. Para comprobar que la GPU está bien configurada se debe ver que al ejecutar en una consola Python las siguientes instrucciones sea devuelto “True”:

```
import torch
torch.cuda.is_available()
```

Sin necesidad de haber realizado el paso anterior, el cual es recomendable, se puede pasar a iniciar el servidor siguiendo estas instrucciones:

1. Descomprimir la carpeta “Servidor”.
2. Abrir el archivo constants.py que se encuentra dentro de la carpeta anterior y modificar los parámetros marcados como “*Database constants*” que permitirán la conexión con tu base de datos.
3. En una consola del sistema operativo dirigirse a la carpeta “Servidor” y ejecutar los siguientes comandos:

```
pip install flask
pip install flask_mysqldb
pip install flask_cors
pip install numpy
pip install sentence_transformers
pip install torch
python serverFlask.py
```

Tras estos pasos el servidor debe estar en funcionamiento y está listo para ejecutar la interfaz web.

Inicialización de la aplicación web

Para la inicialización de la página web es necesario seguir los siguientes pasos:

1. Instalar Node.js y npm. Este último viene incluido en Node.js y se podrá descargar e instalar a través del enlace: <https://nodejs.org/en/download/>.
2. Extraer la carpeta “Web”.
3. Una vez extraída la carpeta con el código, con la consola del sistema operativo acceder a la ubicación de la carpeta y ejecutar los siguientes comandos:

```
npm install  
npm start
```

4. Con ello se abrirá una ventana en nuestro navegador donde se ejecutará la aplicación. Cabe destacar que en el fichero constants.jsx se podrá cambiar la cabecera a las consultas API.

Bibliografía

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser y I. Polosukhin, «Attention Is All You Need,» 2017.
- [2] Sethunya R Joseph, Hlomani Hlomani, Keletso Letsholo, Freeson Kaniwa, Kutlwano Sedimo, «Natural Language Processing: A Review,» 2016.
- [3] Nils Reimers, Iryna Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,» 2019.
- [4] Jeff Johnson, Matthijs Douze, Hervé Jégou, «Billion-scale similarity search with GPUs,» 2017.
- [5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, «Language Models are Few-Shot Learners,» 2020.
- [6] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, Mark Chen, «Hierarchical Text-Conditional Image Generation with CLIP Latents,» 2022.
- [7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, «Evaluating Large Language Models Trained on Code,» 2021.
- [8] D. Chandrasekaran y V. Mago, «Evolution of Semantic Similarity—A Survey,» *ACM Digital Library*, vol. 54, nº 2, Marzo 2022.
- [9] «Wikipedia - TF & TF-IDF,» Wikipedia, 2016. [En línea]. Available: <https://en.wikipedia.org/wiki/Tf%20%93idf>.
- [10] Jeffrey Pennington, Richard Socher, Christopher D. Manning, «GloVe: Global Vectors for Word Representation,» 2014.
- [11] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, «Efficient Estimation of Word Representations in Vector Space,» 2013.

- [12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer, «Deep contextualized word representations,» 2018.
- [13] A. Sherstinsky, «Fundamentals of Recurrent Neural Network (RNN)and Long Short-Term Memory (LSTM) Network,» 2021.
- [14] Guolin Ke, Di He, Tie-Yan Liu, «Rethinking Positional Encoding in Language Pre-Training,» 2021.
- [15] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, «Neural Machine Translation by Jointly Learning to Align and Translate,» 2014.
- [16] «Wikipedia - Feed-forward Neural Networks.,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_prealimentada.
- [17] «Wikipedia - Rectificador en redes neuronales.,» 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_prealimentada.
- [18] Mor Geva, Roei Schuster, Jonathan Berant, Omer Levy, «Transformer Feed-Forward Layers Are Key-Value Memories,» 2020.
- [19] Kishore A Papineni, Salim Roukos, Todd Ward, Weijing Zhu, «BLEU: a method for automatic evaluation of machine translation,» 2002.
- [20] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le, «XLNet: Generalized Autoregressive Pretraining for Language Understanding,» 2020.
- [21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» 2018.
- [22] Wikipedia, «Wikipedia - Siamese neural network,» 2019. [En línea]. Available: https://en.wikipedia.org/wiki/Siamese_neural_network.
- [23] Faisal Rahutomo, Teruaki Kitasuka & Masayoshi Aritsugi , «Semantic Cosine Similarity,» 2012.
- [24] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, Ming Zhou, «MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,» 2020.

- [25] Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, «Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks,» 2020.
- [26] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, Iryna Gurevych, «BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models,» 2021.
- [27] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,» 2019.
- [28] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, Ray Kurzweil, «Efficient Natural Language Response Suggestion for Smart Reply,» 2017.
- [29] Kexin Wang, Nandan Thakur, Nils Reimers, Iryna Gurevych, «GPL: Generative Pseudo Labeling for Unsupervised Domain Adaptation of Dense Retrieval,» 2021.
- [30] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, Allan Hanbury, «Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation,» 2020.
- [31] «iartificial,» [En línea]. Available: <https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/>.
- [32] Laila Rasmy, Yang Xiang, Ziqian Xie, Cui Tao, Degui Zhi, «Med-BERT: pre-trained contextualized embeddings on large-scale structured electronic health records for disease prediction,» 2020.