UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Mejora de la recomendación de vídeos de mediaUPV

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Micó Ibáñez, Alejandro Tobías

Tutor/a: Julian Inglada, Vicente Javier

Director/a Experimental: JORDAN PRUNERA, JAUME MAGI

CURSO ACADÉMICO: 2021/2022

Resum

La quantitat de vídeos en mediaUPV augmenta cada dia a causa de l'auge de la docencia flip i com a conseqüència de la Covid. En aquest treball es planteja investigar tècniques més noves de preprocessament de text, centrant-nos concretament en BERT i en Sentence Transformers, per a poder millorar l'extracció de característiques i, per tant, el desenvolupament de noves tècniques de recomanació per a així millorar la precisió de les recomanacions, millorant a la vegada la qualitat de vida dels usuaris de la aplicació. Finalment, realitzar la validació del funcionament de les noves tècniques i una experimentació per a comparar resultats amb la versió anterior del recomanador.

Paraules clau: Recomanació, aprenentatge automàtic, llenguatge, model

Resumen

La cantidad de vídeos en mediaUPV aumenta cada día debido al auge de la docencia flip y como consecuencia de la Covid. En este trabajo se plantea investigar técnicas más novedosas de preprocesamiento de texto, más concretamente BERT y Sentence Transformers, para poder mejorar la extracción de características y, por tanto, el desarrollo de nuevas técnicas de recomendación para así mejorar la precisión de las recomendaciones, mejorando también la calidad de vida de los usuarios de la aplicación. Por último, realizar validación del funcionamiento de las nuevas técnicas y experimentación para poder comparar resultados con la versión anterior del recomendador.

Palabras clave: Recomendación, aprendizaje automático, lenguaje, modelo

Abstract

The number of videos in mediaUPV is increasing every day due to the rise of flip teaching and as a consequence of Covid. This work deals with the idea to investigate newer text pre-processing techniques, more specifically BERT and Sentence Transformers, in order to improve feature extraction and, therefore, the development of new recommendation techniques to improve the accuracy of the recommendations, also improving the quality of life of the users of the application. Finally, validation of the operation of the new techniques and experimentation in order to compare results with the previous version of the recommender.

Key words: Recommendation, machine learning, language, model

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La reciente pandemia ha afectado mucho a nuestras vidas de muchas formas, tanto en el ámbito social como en el económico pero también en el educativo. En 2020 se tuvo que adoptar una docencia totalmente remota, por lo que muchas de las tecnologías relacionadas tenían que desarrollarse para adaptarse a los cambios. Una de las tecnologías que más auge tuvo en la Universitat Politècnica de València fue mediaUPV ya que el tráfico de la misma aumentó drásticamente al tener que ver vídeos en remoto desde casa. Aunque ya se usaba el sistema, no se había usado a este nivel y surgieron necesidades de mejora. Una de estas mejoras fue la recomendación de los vídeos para los usuarios, implementada hace más de un año. En este proyecto se va a revisar esa misma mejora que se realizó para intentar perfeccionar esta tecnología con los últimos avances en el procesamiento del lenguaje natural.

1.1 Motivación

La motivación de este proyecto surge de la idea de intentar mejorar el recomendador de vídeos del portal de vídeos de la UPV, mediaUPV, con el objetivo de mejorar las recomendaciones que reciben los usuarios. Actualmente el recomendador es el publicado en el artículo de Jordán et al. [2], en 2021 y se quiere mejorar con las últimas tecnologías en la materia para que las recomendaciones sean más eficientes para los usuarios que consumen la aplicación.

1.2 Objetivos

Las metas a lograr para el proyecto son:

- Investigar las diferentes opciones de nuevas tecnologías e intentar elegir la mejor a la hora de implementar las mejoras al recomendador.
- Mejorar el recomendador aplicando técnicas más novedosas de preprocesado de la información e implementar un método nuevo de recomendación que obtenga mejores resultados.

- Comparar las técnicas de preproceso de texto para extraer las características, principalmente TF-IDF, Word2Vec y BERT, analizando los resultados después de probarlas.

1.3 Estructura de la memoria

En esta sección se va a desarrollar el índice antes indicado, explicando punto por punto todos los apartados de la memoria. Después ya nos centraremos en la explicación e implementación de lo investigado y por último hacer pruebas para validar los cambios realizados. La explicación capítulo a capítulo es la siguiente:

- Capítulo 2. Estado del arte: En este capítulo hablaremos sobre la situación actual de las tecnologías utilizadas o relacionadas con el trabajo que se está realizando. Es prácticamente el punto más importante ya que está muy relacionado con la investigación que se realizó para poder implantar luego el código.
- Capítulo 3. Análisis de la situación: En el análisis de la situación explicaremos el estado actual del recomendador, además de revisar cómo está implementado actualmente y revisaremos el conjunto de datos que tenemos para poder hacer pruebas.
- Capítulo 4. Desarrollo de la solución: En este apartado veremos las primeras ideas que se pensaron y los cambios que se han ido realizando finalmente, pero también los problemas que se dieron y las soluciones que le pudimos dar a estos problemas, además de cómo se pudo implementar en Python la solución.
- Capítulo 5. Pruebas: En el capítulo de pruebas, como bien dice su nombre, se hablará de los tests y experimentos que se realizaron para probar que la solución funcionaba, además de poder compararlo con la anterior versión del recomendador. Se hablará además de los tiempos de cada uno de las dos versiones del recomendador.
- Capítulo 6. Conclusiones: En este último capítulo se hablará sobre los resultados del trabajo, hablando sobre qué ha ido bien y sobre qué ha ido mal. Se mencionarán también las materias de la carrera que más están relacionadas con el tema del proyecto y además se hablará sobre posibles futuros proyectos, dando ideas para intentar mejorar el recomendador.
- Anexo A. ODS: Los Objetivos de Desarrollo Sostenible es el primero de los anexos, en el que se desarrollará cuales de los 17 objetivos están más relacionados con el proyecto.
- Anexo B. Glosario: En este último anexo se definirán términos utilizados en la memoria y que pueden ser desconocidos para alguno de los lectores.

CAPÍTULO 2

Estado del arte

En este capítulo se analizarán y explicarán conceptos básicos de requerimiento previo y relacionados al problema al cual nos enfrentamos. Se hablará del procesamiento del lenguaje natural, los sistemas de recomendación e introduciremos herramientas que se han usado para la realización del trabajo. En un primer lugar introduciremos el entorno sobre el que se aplicará el recomendador que es el portal de objetos de aprendizaje en forma de vídeos de la UPV, conocido como mediaUPV.

2.1 MediaUPV

MediaUPV [3] es el portal de vídeos de la Universitat Politècnica de València (UPV). Los profesores y alumnos pueden subir vídeos a esta plataforma y el resto de usuarios del servicio puede visualizarlos. El sistema también dota de transcripción y traducción automáticas gracias a transLectures [4].

Figura 2.1: Ejemplo de vídeo en mediaUPV

Con la reciente pandemia, el número de vídeos publicados ha aumentado considerablemente, gracias a las clases en remoto y el aumento de interés en la plataforma. Esto ha conllevado a que sea más difícil para los usuarios el encontrar contenido relevante. Para solucionar esto, el servicio tiene activo un recomendador de vídeos que combina las técnicas basadas en el contenido (content-based), obtenido a partir de las transcripciones, con el collaborative-filtering.

Como vemos en la figura 2.1, los vídeos relacionados salen a la derecha del mismo vídeo, de manera similar a otras plataformas de vídeo como por ejemplo YouTube.

2.2 Conceptos importantes

En esta sección se verán algunos conceptos que se tendrán que entender para poder comprender los siguientes puntos.

2.2.1. Machine Learning

Es una disciplina del campo de la Inteligencia Artificial y la Informática que se basa en usar datos y algoritmos para predecir y aprender de forma similar a los humanos, mejorando gradualmente [5]. En los últimos años se está desarrollando de formas cada vez más impresionantes.

Hay técnicas con aprendizaje automático en muchos campos de la ciencia, como la visión por ordenador, reconocimiento de formas, voces y texto, aeronáutica e incluso tiene aplicaciones médicas. Es decir, no es algo inusual, sino cada vez más frecuente en nuestras vidas.

2.2.2. Redes neuronales

Una red neuronal artificial, o solo red neuronal [6], es básicamente el intento de representar e imitar una red neuronal humana en un computador. Es decir, intentar reproducir las conexiones entre los nodos y sus comunicaciones de manera que sean de forma similar a cómo se comunican las neuronas entre sí.

Lo más llamativo de éstas es, al igual que en un cerebro humano, el aprendizaje. Las redes "aprenden" ganando conocimiento a partir de una base de datos. Esto es, se le entrena con datos con respuesta y aprenden estos resultados para después deducir las demás respuestas. Este método de enseñanza se denomina comúnmente aprendizaje supervisado y es una forma de aprendizaje automático o Machine Learning.

Las aplicaciones que tienen las redes neuronales son muy variadas y se pueden ver utilizadas por tanto en muchos campos, como pueden ser las matemáticas, con aproximación de funciones o la regresión; en robótica, ingeniería de control o incluso, la que nos interesa para el proyecto en cuestión: procesamiento de datos.

2.3 Procesamiento del lenguaje natural

El lenguaje natural es la manera en la que los humanos nos comunicamos entre nosotros, es decir, el lenguaje oral y escrito.

El procesamiento del lenguaje natural, o NLP por sus siglas en inglés (Natural Language Processing) [7], es la manipulación automática del lenguaje natural. Es una rama de la inteligencia artificial, IA, que se centra en intentar dar a los computadores la capacidad de entender textos y/o palabras habladas como si se tratase de un ser humano.

Combina técnicas estadísticas, de aprendizaje profundo (deep learning) y de aprendizaje automático (machine learning).

Con estas técnicas de preproceso se han conseguido diversas soluciones a problemas, como por ejemplo la traducción de texto, reconocimiento de palabras habladas o sistemas de recomendación, entre otros.

2.3.1. Técnicas de clasificación de texto

En este apartado, se presentan diferentes técnicas de clasificación de texto y sus ventajas y desventajas.

TF-IDF

El TF-IDF (Term Frequency - Inverse Document Frequency) [8] es un término que se refiere a la medida usada para determinar la importancia de cada palabra con respecto a los documentos. El TF se refiere a la cantidad de repeticiones de la palabra en un documento y el IDF a la frecuencia inversa de la palabra en todos los documentos. El producto de ambos nos da el TF-IDF.

En este ejemplo podemos ver como calcularlo, teniendo un documento A: “El río suena cuando agua lleva” y un documento B “El agua cae cuando llueve mucho”:

Tabla 2.1: Ejemplo de cálculo de TF-IDF

Palabras	TF		IDF	TF*IDF	
	Doc. A	Doc. B		Doc. A	Doc. B
El	1/6	1/6	log(2/2) = 0	0	0
Río	1/6	0	log(2/1) = 0.3	0.05	0
Suena	1/6	0	log(2/1) = 0.3	0.05	0
Cuando	1/6	1/6	log(2/2) = 0	0	0
Agua	1/6	1/6	log(2/2) = 0	0	0
Lleva	1/6	0	log(2/1) = 0.3	0.05	0
Cae	0	1/6	log(2/1) = 0.3	0	0.05
Llueve	0	1/6	log(2/1) = 0.3	0	0.05
Mucho	0	1/6	log(2/1) = 0.3	0	0.05

Así vemos que los elementos irrelevantes son los que se repiten en los dos documentos, por tanto obtienen puntuación de 0, mientras que los que tienen valor distinto a 0 son relevantes. En documentos más largos veríamos valores distintos y podríamos ordenar de mayor a menor importancia en cada documento.

Su uso en Machine Learning está centrado en extracción de palabras clave y recuperación de información. El primero de ellos simplemente se basa en seleccionar las palabras con mayor valor de TF-IDF, que serán las palabras más representativas de ese documento comparado con los otros. El segundo se puede usar para buscadores y para encontrar resultados con más relevancia. Por ejemplo si se busca en una base de datos con varios documentos "Machine Learning", nos saldrán como documentos más relevantes los relacionados con el aprendizaje automático y la inteligencia artificial, ya que estos documentos tendrán "Machine learning" con un valor alto en su TF-IDF.

Word2Vec

Word2Vec [9] es una técnica usada para el procesamiento del lenguaje natural que está basada en representar un texto con vectores de números reales, teniendo en cuenta contexto, significado y similitud semántica. Esto ayuda a reducir la dimensionalidad de un texto con respecto a otros métodos de procesamiento del mismo.

Para ello, lo que se suele utilizar es el resultado que devuelve una capa densa, que es multiplicar una matriz de embedding por la representación de la palabra.

Hay dos principales algoritmos o variantes de Word2Vec, una que predice una palabra objetivo a partir del contexto, CBOW (Common Bag Of Words)[1], y la otra al contrario, predice las palabras del contexto a partir del objetivo, Skip-Gram[2].

El CBOW dispone de 3 capas, la primera capa convierte cada palabra en un vector de embedding, por eso se suele llamar a esta capa la capa Embedding. La matriz de embedding se va aprendiendo mientras se va entrenando el modelo. La segunda capa sumaría los embeddings y obtendría una dimensión de salida. Por último la capa final se encarga de predecir la palabra objetivo. Este modelo está centrado por tanto en aprender las relaciones sintácticas de las palabras.

El Skip-gram, en cambio, se basa en el camino contrario al CBOW, es decir obtener el contexto a partir de una palabra objetivo. Por tanto las capas de la red son las mismas pero al contrario. Este modelo se centra más en aprender las relaciones semánticas al contrario que el CBOW.

[1] Capítulo 3.1 del paper de Mikolov et al.[9]
[2] Capítulo 3.2 del paper de Mikolov et al.[9]

Figura 2.2: Esquema básico de CBOW

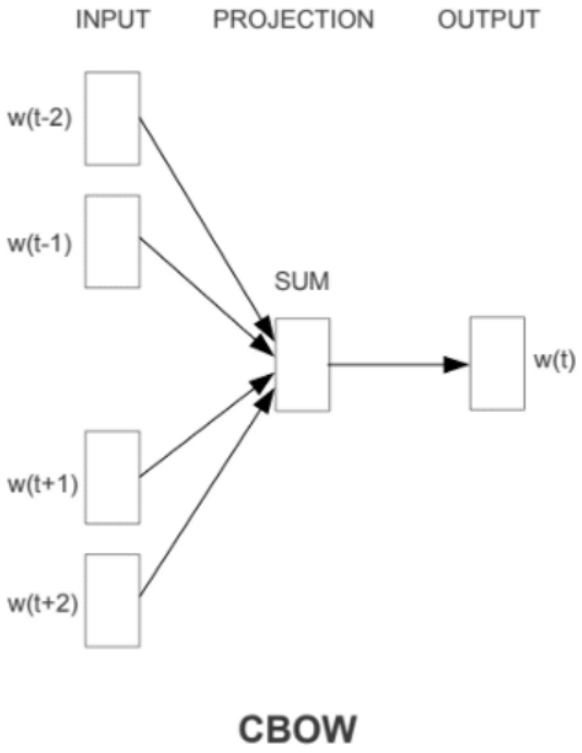
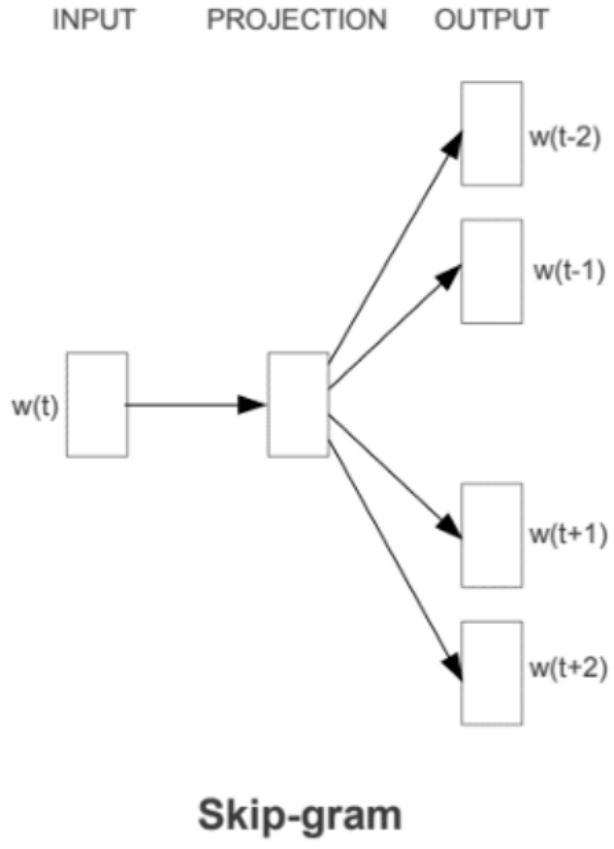


Figura 2.3: Esquema básico de Skip-Gram



De acuerdo a Mikolov et al.[9], el método de las Skip-Grams funciona mejor para conjuntos de datos pequeños y representa mejor palabras que son menos frecuentes, mientras que CBOW mejora cuando se trata de entrenar de forma rápida y representa mejor, en cambio, palabras más frecuentes.

A continuación hablaremos de algunos ejemplos que se mencionan en el artículo sobre el modelo de Skip-Gram, hablando sobre las relaciones entre palabras. Ponen de ejemplo que *París - Francia + Italia = Roma*, que se obtiene a partir de la relación que tienen Francia y París de ser país y capital, entonces al hacer París - Francia y añadiendo otro país, en este caso Italia, se queda con que quiere la capital de Italia y devuelve Roma. En el artículo se mencionan más relaciones como profesiones, presidentes, comidas típicas, etc.

La conclusión es que es un algoritmo muy útil ya que analiza las propiedades tanto semánticas como sintácticas de la palabra, ya que se basa en contexto y por tanto reduce el tamaño de la dimensionalidad.

2.3.2. Modelos del lenguaje

Un modelo de lenguaje es básicamente una distribución de probabilidades sobre las palabras o secuencias de palabras. El modelo da la probabilidad de ser válida en el contexto de ser similar al lenguaje natural. El objetivo principal es el entendimiento del lenguaje natural para distintas tareas o propósitos. Los modelos se basan en calcular las probabilidades de n-gramas (n siendo número de palabras), por lo que trae problemas cuando se trata de textos complicados, ya que en ocasiones necesitarías n-gramas con n muy alta para que el modelo "entienda" el contexto. El problema del contexto lleva a que cuanto más alta sea la n, las posibles combinaciones de palabras aumentan, llevando a que las probabilidades estén más repartidas, aunque contengan palabras muy comunes en el texto.

Recientemente, han aparecido modelos basados en redes neuronales, que solucionan el problema de las probabilidades ya que tratan el texto de otra manera. Estos modelos se caracterizan en que tienen aprendizaje, lo que consigue reducir la dimensionalidad, es decir, el problema de la necesidad de grandes cantidades de datos para poder extraer resultados significativos.

BERT

BERT [1], o Bidirectional Encoder Representations from Transformers, es un modelo de lenguaje de código abierto usado para el procesamiento del lenguaje natural. BERT aplica el entrenamiento bidireccional de Transformer, un modelo que adopta el mecanismo de la auto-atención, al modelado del lenguaje.

Transformer aprende relaciones contextuales entre palabras en un texto. Por sí mismo incluye un codificador que lee el input del texto y un decodificador que produce predic-

ciones. Es bidireccional, esto es, que no lee el texto de forma secuencial sino todo a la vez, lo que le permite aprender el contexto de la palabra.

BERT se suele utilizar para muy diversas funciones tales como la predicción de la siguiente palabra o frase, en QA (Question Answering), reconocimiento de entidades, etc. Pero lo más importante es que se puede adaptar a casi cualquier tipo de NLP para el que lo necesites.

BERT requiere de dos etapas, una de preentrenamiento y otra de fine-tuning. La primera es la más costosa y puede llevar días, pero solo hay que hacerlo una vez por modelo o lenguaje dentro del modelo (para los multilengua). La fase de fine-tuning no es tan costosa computacionalmente y puede llevar como máximo unas horas en una GPU.

Dispone de varios modelos preentrenados, es decir, modelos con conocimientos sobre el lenguaje natural. Hay de varios tipos, se pueden distinguir entre *Cased* y *Uncased*, que se diferencian en que el *Uncased* convierte el texto en minúsculas y quita acentos y similares, mientras que el *Cased* deja el texto sin modificar. Además están los de una lengua concreta, sobretodo en inglés aunque podemos encontrar de varios idiomas, y los multilengua, que suelen ser peores que los de una lengua en concreto pero son útiles para textos multiidioma o para ciertos idiomas en concreto.

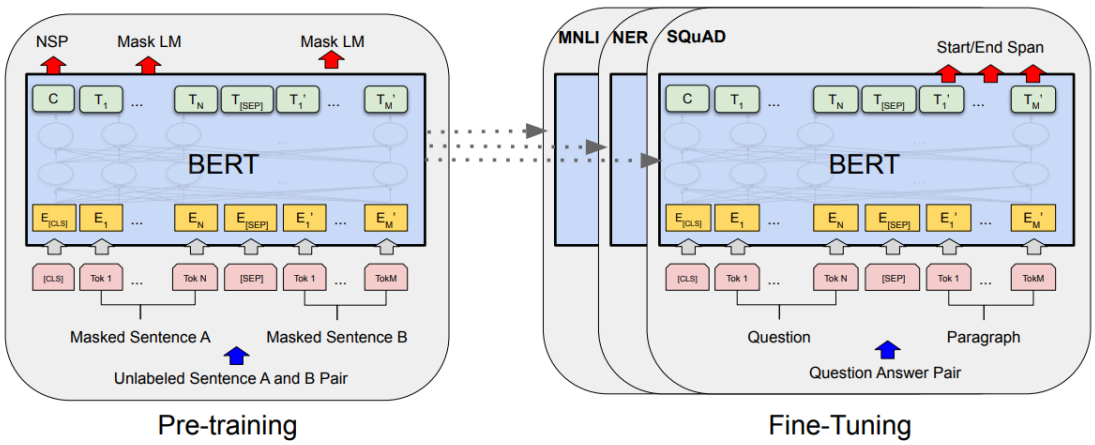
En vez de hacer el fine-tuning, en algunas situaciones querremos obtener los embeddings (representaciones contextuales de cada token de entrada generado por el modelo preentrenado). Para ello usaremos el tokenizer, que normalizará el texto, separará los caracteres de puntuación y convertirá en tokens el resto de caracteres. Por ejemplo, al introducir el texto:

Peter Peterson's dog.

Lo transformará en:

Peter ∣ Peter ∣ ##son ∣ ' ∣ s ∣ dog.

Figura 2.4: Esquema básico de BERT [1]



En la figura 2.4, podemos ver como funcionan los métodos de preentrenamiento y de fine-tuning, en este caso para SQuAD.

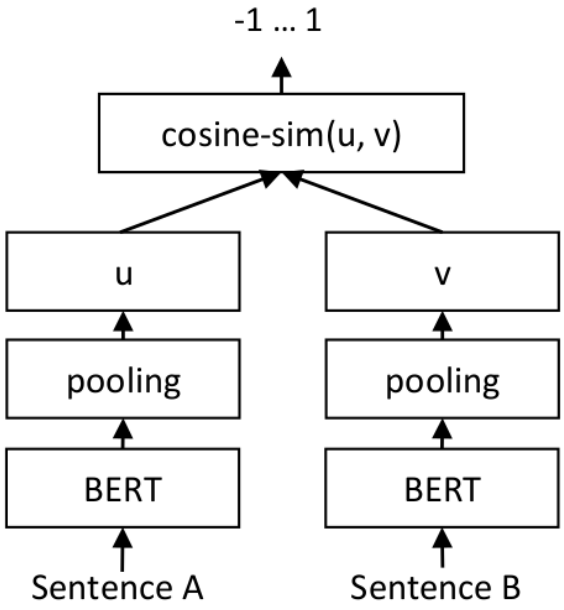
En conclusión, BERT es una herramienta que promete mucho en cuanto a la innovación sobre el preprocesamiento del lenguaje natural y que puede resultar muy útil si se utiliza correctamente.

Sentence-BERT

Sentence BERT o Sentence Transformers [10], es un modelo basado en BERT, que puede computar frases o textos típicamente más largos que BERT y de forma más eficiente, ya que es un modelo pensado para ello.

La forma de usarlo se simplifica mucho con respecto a la librería de Transformers básica ya que es más específico para el proceso de frases.

Figura 2.5: Esquema simplificado de funcionamiento de S.Transformers



2.4 Sistemas de Recomendación

Un sistema de recomendación es un algoritmo o herramienta diseñado para sugerir algo a un usuario en concreto basándose en distintos factores.

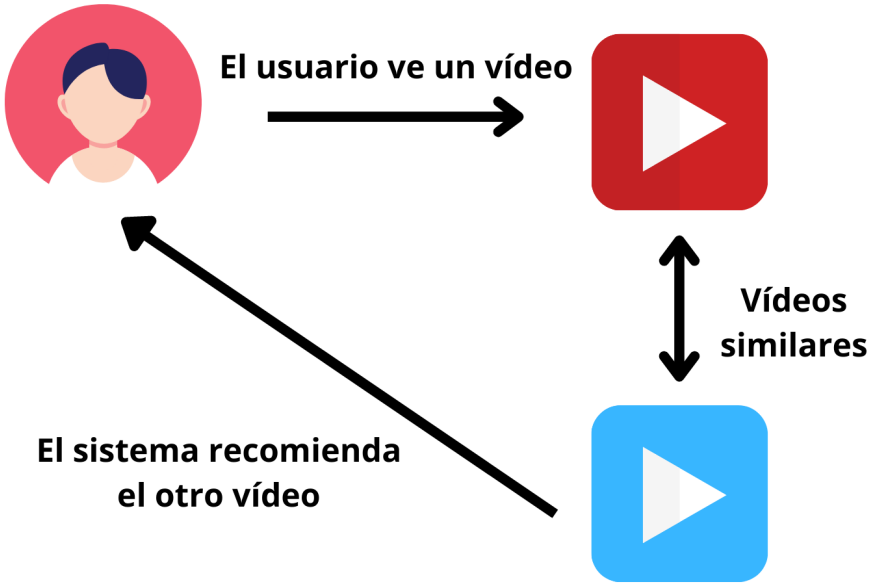
Hay de varios tipos:

- Sistemas content-based o basados en el contenido, que se basan en el perfil del usuario en cuanto a sus preferencias con respecto a las distintas opciones.
- Collaborative Filtering, que se basan en recomendar al usuario en base a lo que les ha gustado a usuarios similares a este.
- Híbridos, que combinan los dos anteriores para una mayor efectividad.

2.4.1. Content-based

Los sistemas basados en el contenido o content-based [11], se centran en el contenido de los ítems para buscar perfiles parecidos entre ellos. Por ejemplo en textos, que tengan palabras clave parecidas. Entonces el funcionamiento es que cuando un usuario ha visto un vídeo, por ejemplo, le recomiende otro vídeo cuyo contenido es similar, basado en distintos factores de su contenido, en este caso podría ser título, autor o incluso contenido, extrayendo su transcripción.

Figura 2.6: Esquema sencillo de content-based filtering



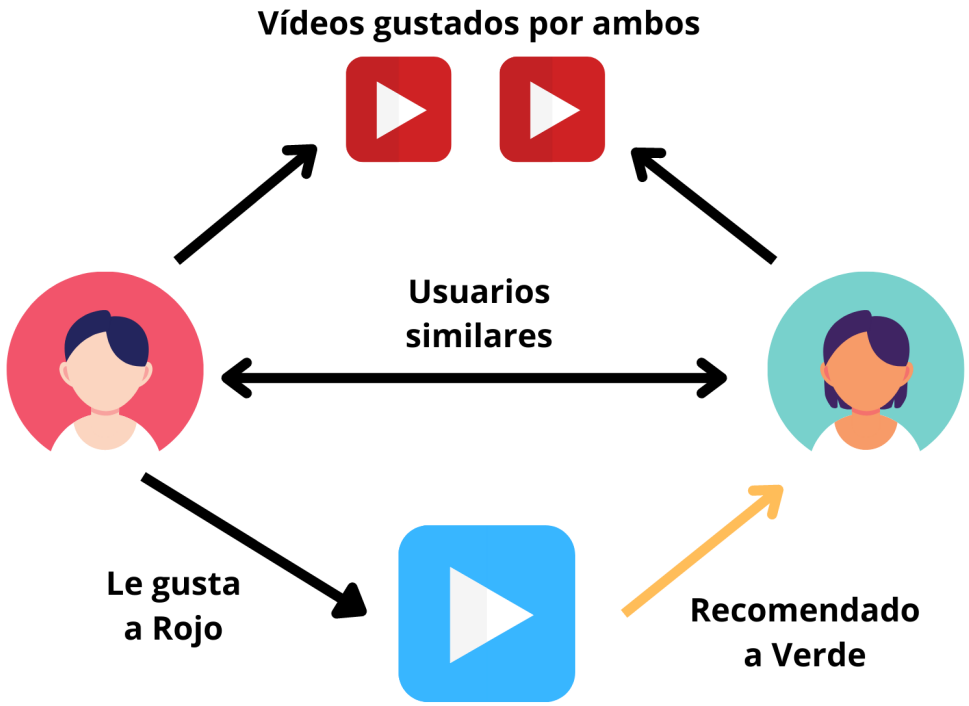
En la figura 2.6 podemos ver que se recomiendan items similares a los vistos por el usuario. Cuando el usuario ha visto el vídeo, y con todo su perfil de vídeos vistos, el sistema busca vídeos similares a los que están en su lista de vídeos vistos y recomienda los que más puntuación de similitud tengan.

El filtrado content based puede ser útil en situaciones donde las recomendaciones se centren en una persona en concreto sin necesidad de saber datos sobre el resto de usuarios, por tanto no se necesita que un ítem sea valorado por varios usuarios para poder recomendarlo. Las desventajas que supone este método serían que se necesita mucha información para llevar a cabo las recomendaciones y que solo puede dar recomendaciones de los intereses del usuario y no de otros temas.

2.4.2. Collaborative Filtering

El collaborative filtering [12] es una técnica que se usa en los sistemas de recomendación que consiste en hacer predicciones gracias a las preferencias de muchos usuarios. Se basa en que si a un sujeto A comparte opinión en algo con B, es más probable que coincidan en otras opiniones. Como vemos en el siguiente esquema de una forma simplificada, el sistema recomienda vídeos entre usuarios similares.

Figura 2.7: Esquema sencillo de collaborative filtering



En la figura 2.7, podemos ver que tanto a Alice como a Bob les han gustado ciertos vídeos, por lo que el sistema los reconoce como usuarios similares. Como son similares y a Alice le gusta un vídeo, el sistema se lo recomienda a Bob. El collaborative filtering

también se puede dar basado en los ítems en vez de en los usuarios, esto es, en vez de considerar usuarios similares para recomendar entre ellos, buscaría ítems similares para recomendar otros. Por ejemplo en el contexto del ejemplo anterior, los vídeos que han visto tanto Alice como Bob serían ítems similares, así que a los usuarios que han visto uno de esos vídeos podrían recomendarle el otro.

CAPÍTULO 3
Análisis de la situación

Con la gran crecida que tuvo en consecuencia de la COVID-19 y la obligatoriedad de tener que realizar clases online, además del auge que ya estaba teniendo el online teaching, el portal de mediaUPV cada día tiene más vídeos, actualmente alrededor de 92.000 (en agosto de 2022). La mayoría de los vídeos tienen transcripción gracias al generador automático.

Actualmente el recomendador es un sistema híbrido que junta los métodos de content-based y de collaborative filtering para buscar los vídeos a recomendar. El código que se utiliza para llevar a cabo el proceso fue desarrollado por Jordán et al [2]. Está escrito en Python ya que es el más cómodo para realizar este tipo de código.

Las librerías que utiliza son las básicas de Python además de algunas útiles para el propio TF-IDF. Las más destacables son:

- Paquete nltk: Este paquete es el que permite trabajar con el lenguaje natural en Python, conteniendo diversos métodos de clasificación, tokenización, stemming, etc., además de otras utilidades. Se usa también para descargar la lista de stopwords que se usará en la técnica del TF-IDF.
- Paquete sklearn: Usado para las métricas, pero también para el vectorizer del TF-IDF y para métodos para calcular el Cosine Similarity o el Nearest Neighbors. El Cosine Similarity o la similaridad coseno es lo que usamos para ver cuan distinta es una palabra de la otra.
- Paquete loguru: El paquete se importa para tener el logger, que es lo que se encarga de las impresiones para ver el estado de la ejecución, además de poder debuggear el código si fuese necesario.
- Paquetes varios tales como scipy, tqdm, numpy, pandas, etc., con funciones variadas útiles para tratamiento de datos y fórmulas diversas.

El archivo principal es *recommender.py*, que es la clase que contiene todos los métodos necesarios para hacer las recomendaciones. Los métodos principales son los de recommendation que devuelven una lista de recomendaciones dependiendo de los parámetros

que se les pase (la matriz de similaridad de los vídeos, la id del usuario, la lista total de los vídeos y el número de recomendaciones). Además de estos, tenemos los métodos auxiliares que hacen subfunciones para estos métodos de recomendación. Los más importantes son el *item_profile* y el user profile junto con el calculate data recommendation, que son los que calculan la matriz de similitud entre los vídeos.

El archivo *update.py* es el que se ejecuta para recalcular las recomendaciones, es un archivo que se ejecuta todas las noches mediante un script que se corre en el server.

El archivo *test.py* es con el que se pueden ejecutar todas las pruebas necesarias para analizar resultados y así decidir alternativas para el recomendador final.

La colección de datos con la que se ha trabajado son un conjunto de ficheros con las siguientes características:

- Ficheros con un diccionario que contiene las transcripciones y los títulos de los vídeos asociados a su id. Se ha trabajado con un fichero, de nombre *df_videos_full* con las transcripciones de los vídeos hasta agosto de 2019. Tenemos en el conjunto de datos un total de 13232 vídeos con transcripciones.

- Ficheros con la actividad de los usuarios en un período de tiempo. Es un diccionario con cada ítem siendo un id de un vídeo, un id de un usuario, la id de la sesión y la fecha en la que se vio el vídeo. De estos ficheros se usaban como datos de entrenamiento la actividad de septiembre de 2018 hasta abril de 2019 y de test los datos de mayo a julio de 2019. Además luego se probó a juntar ambos ficheros para separar los datos en training y test de manera aleatoria y llamando al fichero complete.

  Estos ficheros se diferenciaban en varios conjuntos de datos a su vez, cogiendo a todos los usuarios, cogiendo solo usuarios que han visto de 1 a 9 vídeos, es decir, a los usuarios nuevos y otros cogiendo a los usuarios que han visto de 10 a 150 vídeos, que son los usuarios que han visto un número de vídeos aceptable. Los usuarios con más de 150 vídeos se ignoran ya que suelen ser usuarios del sistema y otros que manchaban los resultados obtenidos. Por tanto el nombre de estos ficheros es de (mes inicio actividad-mes final actividad) + Act + “1-9” // “1-150” // “” (si es contando todos los usuarios), por ejemplo: 2018-9-2019-4-Act1-9.

- Además contábamos con ficheros con las ids de tanto los vídeos como los usuarios por separado.

- Ficheros rec: resultados de realizar tests sobre el recomendador, conteniendo para cada id de vídeo e id de usuario un conjunto de recomendaciones. Usado para comparar las distintas pruebas que se realizan y así ver los resultados de los experimentos realizados. Los ficheros rec se pueden leer y analizar mediante el archivo *process_reco_results.py*, encargado de mostrar las estadísticas para poder ver cuan satisfactoria ha sido la recomendación a la hora de hacer tests.

CAPÍTULO 4

Desarrollo de la solución propuesta

En este capítulo veremos el desarrollo que se ha seguido para poder llegar al resultado final y todos los pasos que se han llevado a cabo para ello, además de las dificultades encontradas previamente y durante el mismo.

4.1 Primeras ideas

El objetivo de este trabajo era analizar alternativas posibles al método utilizado actualmente para el recomendador, para intentar mejorar su eficiencia. Nuestras alternativas principales para comparar con el actual TF-IDF que estaba implementado eran el word embedding, el word2vec y BERT.

Decidimos desde el principio que BERT podría ser la mejor opción ya que era la tecnología más novedosa y que más podría mejorar la recomendación, así que todos los esfuerzos de las primeras semanas fueron sobre investigar las tecnología y sobre cómo sería posible implementar en el código.

4.2 Dificultades durante el desarrollo

Durante la realización de este proyecto surgieron varios problemas, primeramente porque el código no era tan fácil de entender sin explicaciones. La parte buena es que el director experimental del proyecto es quien realizó el código en su momento, así que explicó cualquier duda que hubiera surgido.

Hubo dificultades a la hora de investigar, ya que por un lado no había mucha información sobre el uso de BERT para el tema que necesitábamos y además, dado que era un software que no se ha visto durante la carrera y también debido a que la curva de aprendizaje del mismo es muy alta, el tiempo dedicado a la investigación y al entendimiento de BERT fue más del esperado.

Después de haber investigado suficiente, a la hora de poner lo aprendido sobre BERT/BETO en práctica surgieron varios inconvenientes relacionados con la potencia máxima del PC. A la hora de realizar un test, el programa consumía la memoria máxima del

ordenador y no se podía realizar ninguna acción además de ejecutar las pruebas. Esto sumado a la larga duración de las pruebas y a los fallos que dieron hasta que se consolidó el código correctamente, gastó muchas más horas de las previstas.

4.3 Implementación del código

Para poder implementar el código con los nuevos cambios previstos se necesitaron instalar varios paquetes:

- Paquete de Torch: La librería Torch[13] incluye todo lo necesario para controlar tensores multidimensionales y utilidades para serializarlos, además de otros usos útiles para aprendizaje automático e IA. Usamos esta en conjunto con la de Transformers para poder usar métodos de BERT a la hora del preproceso del texto. Antes de decantarnos por Torch planeamos en usar TensorFlow, pero encontramos un ejemplo de uso de BERT que usaba precisamente los pesos de PyTorch así que al final lo cambiamos en lugar de usar TensorFlow. De haberlo usado, habría que haber instalado previamente tanto el paquete básico de Tensorflow, como el de Tensorflow Hub.

- Paquete Transformers: Transformers[14] es una librería de Python utilizada para poder descargar, entrenar y usar modelos del lenguaje preentrenados. Los modelos se pueden usar para distintas finalidades tales como reconocimiento y clasificación de imágenes, de audio, de respuesta a preguntas, etc, además de clasificación de texto que es lo que nos incumbe. La Librería tiene la posibilidad de elegir entre las tres librerías más conocidas de aprendizaje profundo, estas son, PyTorch, TensorFlow y JAX. Nosotros usaremos PyTorch al final. Es la librería que necesitaremos obligatoriamente para poder usar BERT.

- Sentence Transformers: La librería de Sentence Transformers tiene lo necesario para poder ejecutar funciones de estos modelos del lenguaje. Es la necesaria para poder usar el modelo de lenguaje que se utilizó al final.

La primera idea que intentamos llevar a cabo fue el implementar BERT pero con el corpus en español llamado BETO. BETO[15] es un modelo del lenguaje basado en BERT que se preentrenó con un corpus que incluye fuentes tales como wikis en Español (Wikipedia, Wikinews, Wikiquotes...), secciones de páginas en castellano (ParaCrawl, EUBookshop, MultiUN, etc.) y otras como OpenSubtitles o GlobalVoices, TED e incluso de las Naciones Unidas. Gracias a esto, el número de tokens que tiene es de alrededor de 3 billions (3 mil millones).

BETO tiene resultados sobre textos en español muy convincentes comparándose al multilingual con mejores resultados que tiene el modelo de BERT, como podemos observar en la tabla que muestran en su GitHub[16]:

Figura 4.1: Tabla comparativa BERT vs BETO cased y uncased, y otros resultados

Task	BETO-cased	BETO-uncased	Best Multilingual BERT	Other results
POS	**98.97**	98.44	97.10 [2]	98.91 [6], 96.71 [3]
NER-C	**88.43**	82.67	87.38 [2]	87.18 [3]
MLDoc	95.60	**96.12**	95.70 [2]	88.75 [4]
PAWS-X	89.05	89.55	90.70 [8]	
XNLI	**82.01**	80.15	78.50 [2]	80.80 [5], 77.80 [1], 73.15 [4]

Durante la investigación para ir probando todos los métodos que íbamos encontrando e ir aprendiendo sobre ellos se creó un archivo notebook de Python. En él se fueron probando alternativas para ir testeando qué método era mejor para la función que queríamos que desempeñara el método sin tener que implementarlo del todo en el código. De hecho, en esta notebook es donde se probó el primer tokenizer que pudimos implementar, uno muy sencillo con tan solo 4 frases y que las comparaba entre ellas. Es lo que nos ayudó a entender el funcionamiento básico que seguía BERT para hacer sus cálculos y luego poder implementar lo mismo en el código.

Se intentó implementar BERT usando la librería anteriormente mencionada Transformers para cambiar lo que hacía el método del TF-IDF para que lo realizara BERT. Para ello creábamos el modelo y el tokenizer mediante los métodos establecidos para ello: *BertTokenizer.from_pretrained()* y *BertForMaskedLM.from_pretrained()* y pasando a ambos como parámetro la carpeta donde estaban los pesos de PyTorch así como la configuración del modelo.

Figura 4.2: Tokenizer de BERT

class transformers.BertTokenizer <source>

```
( vocab_file, do_lower_case = True, do_basic_tokenize = True, never_split = None, unk_token =
'[UNK]', sep_token = '[SEP]', pad_token = '[PAD]', cls_token = '[CLS]', mask_token = '[MASK]',
tokenize_chinese_chars = True, strip_accents = None, **kwargs )
```

Figura 4.3: Método from pretrained de BERT

```
from_pretrained                                                    <source>

( pretrained_model_name_or_path: typing.Union[str, os.PathLike, NoneType], *model_args, **kwargs
)
```

Justo después se evaluaba el modelo y se hacía un encode de cada texto de cada vídeo con el método de *tokenizer.encode_plus()* con parámetros el texto, eligiendo si se trunca, el padding y escogiendo los tensores de retorno de PyTorch con 'pt'. Usamos el encode plus porque realiza todas las funciones que tendríamos que hacer por separado automáticamente, estas son, separar en tokens, añadir los tokens especiales, mapear cada token con su id, truncar los textos para que tengan igual longitud y por ultimo crear las máscaras de atención. Los resultados de cada se añadían a un diccionario de *input_ids* y de *attention_mask*. Cuando se procesan todos los textos en el diccionario de tokens se pasan al modelo para sacar los outputs, de ahí sacamos los embeddings y gracias a la attention_mask y estos últimos sacamos los masked_embeddings que se normalizan. Se hacía lo mismo con los títulos y se juntaban con los textos. Esto es el item profile, después se trataba de igual forma que con el TF-IDF.

Figura 4.4: Método encode plus de BERT

```
encode_plus                                                        <source>

( text: typing.Union[str, typing.List[str], typing.List[int]], text_pair: typing.Union[str,
typing.List[str], typing.List[int], NoneType] = None, add_special_tokens: bool = True, padding:
typing.Union[bool, str, transformers.utils.generic.PaddingStrategy] = False, truncation:
typing.Union[bool, str, transformers.tokenization_utils_base.TruncationStrategy] = False,
max_length: typing.Optional[int] = None, stride: int = 0, is_split_into_words: bool = False,
pad_to_multiple_of: typing.Optional[int] = None, return_tensors: typing.Union[str,
transformers.utils.generic.TensorType, NoneType] = None, return_token_type_ids:
typing.Optional[bool] = None, return_attention_mask: typing.Optional[bool] = None,
return_overflowing_tokens: bool = False, return_special_tokens_mask: bool = False,
return_offsets_mapping: bool = False, return_length: bool = False, verbose: bool = True, **kwargs
) → BatchEncoding
```

Después de los cambios necesarios para que el programa funcionara sin fallos, el código era el siguiente:

```
def item_profile_bert(self, videos_file_name,
                      num_features_transcription=1000,
                      num_features_title=150):
    df_videos = self.read_dataframe(videos_file_name)
    df_videos = df_videos.sort_values(by=['_id'])
```

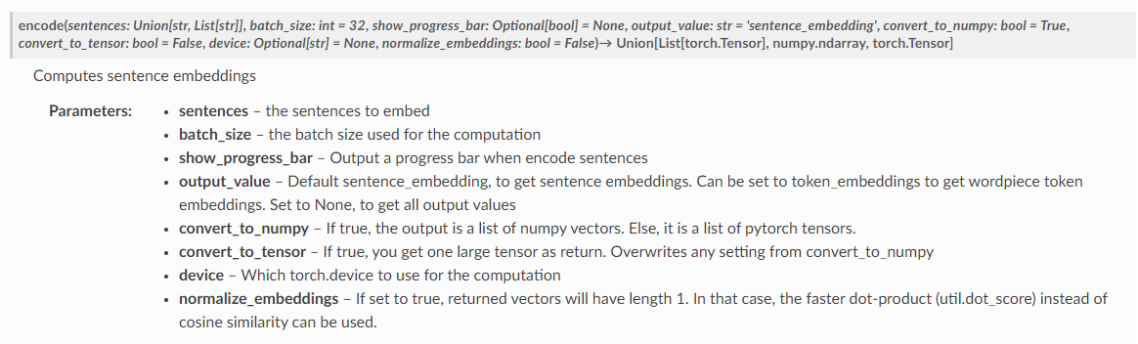
```
    tokenizer = bert.BertTokenizer.from_pretrained("pytorch/",
                                        do_lower_case=False)
model = bert.BertForMaskedLM.from_pretrained("pytorch/")
e = model.eval()
corpus = df_videos["transcription"][0:100]
tokens = {'input_ids': [], 'attention_mask': []}
for sentence in corpus:
    new_tokens = tokenizer.encode_plus(sentence,
                            max_length=num_features_transcription,
                            is_split_into_words=True,
                            truncation=True, padding='max_length',
                            return_tensors='pt')
    tokens['input_ids'].append(new_tokens['input_ids'][0])
    tokens['attention_mask'].append(new_tokens['attention_mask'][0])
tokens['input_ids'] = torch.stack(tokens['input_ids'])
tokens['attention_mask'] = torch.stack(tokens['attention_mask'])

outputs = model(**tokens)
embeddings = outputs.logits
attention_mask = tokens['attention_mask']
mask = attention_mask.unsqueeze(-1).expand(embeddings.size()).float()
masked_embeddings = embeddings * mask
summed = torch.sum(masked_embeddings, 1)
summed_mask = torch.clamp(mask.sum(1), min=1e-9)
mean_pooled_trans = summed / summed_mask

# convert from PyTorch tensor to numpy array
# mean_pooled = mean_pooled.detach().numpy()
item_profile_bert = mean_pooled_trans.detach().numpy()

return item_profile_bert
```

Al ejecutar el test, la maquina donde se estaba realizando el experimento, siendo un ordenador personal, se quedaba sin memoria RAM para procesarlo todo, pese a tener 32GB, y además tardaba una cantidad de horas que era inviable y, tras muchos intentos de arreglar el código a nuestro favor sin éxito, decidimos buscar alternativas u otros métodos que funcionaran mejor.

Probamos a cambiar al multilingual por si era cosa del modelo de BETO en concreto y aún así no se solucionaba el problema por tanto decidimos que lo más sensato sería cambiarlo por completo. Esto nos retrasó mucho ya que todos los esfuerzos que se habían dedicado a la investigación y sobretodo a la implementación, que llevó bastante tiempo hasta que funcionó se fue casi en vano ya que había que cambiar de modelo enteramente.

En este tiempo de investigación encontramos SentenceTransformers, que parecía que podría solucionar el tema de la memoria ya que usaba un método más sencillo y prometía que estaba especializado en textos. Implementarlo fue más sencillo que BETO, ya que este era más específico y su API es más reducida para el tema en cuestión.

Figura 4.5: Encode de S. Transformers


encode(sentences: Union[str, List[str]], batch_size: int = 32, show_progress_bar: Optional[bool] = None, output_value: str = 'sentence_embedding', convert_to_numpy: bool = True, convert_to_tensor: bool = False, device: Optional[str] = None, normalize_embeddings: bool = False)→ Union[List[torch.Tensor], numpy.ndarray, torch.Tensor]

Computes sentence embeddings

Parameters:
- **sentences** – the sentences to embed
- **batch_size** – the batch size used for the computation
- **show_progress_bar** – Output a progress bar when encode sentences
- **output_value** – Default sentence_embedding, to get sentence embeddings. Can be set to token_embeddings to get wordpiece token embeddings. Set to None, to get all output values
- **convert_to_numpy** – If true, the output is a list of numpy vectors. Else, it is a list of pytorch tensors.
- **convert_to_tensor** – If true, you get one large tensor as return. Overwrites any setting from convert_to_numpy
- **device** – Which torch.device to use for the computation
- **normalize_embeddings** – If set to true, returned vectors will have length 1. In that case, the faster dot-product (util.dot_score) instead of cosine similarity can be used.

Para implementar lo que antes nos había llevado decenas de líneas de código, esta se resumía en menos de 10. Había que crear el modelo y ejecutar el método encode, pasándole antes al modelo cuantas features querías para cada texto. Para ello primero probamos con un modelo multilingual que es el *distiluse-base*, un modelo cased, y para otras pruebas más adelante ya escogimos el modelo de hiiamsid[17], que aumentaba el número de features que cogía el modelo.

```
def item_profile_bert(self, videos_file_name,
        num_features_transcription=512, num_features_title=0):
    df_videos = self.read_dataframe(videos_file_name)

    df_videos = df_videos.sort_values(by=['_id'])

    model=st.SentenceTransformer("hiiamsid/sentence_similarity_spanish_es")
                                    #distiluse-base-multilingual-cased-v1
    model.max_seq_length = num_features_transcription

    transcriptions = df_videos["transcription"].to_list()

    embeddings = model.encode(transcriptions)

    return embeddings
```

Este modelo ya nos servía para poder hacer pruebas ya que el ordenador donde se hacían ya soportaba la carga necesaria para ejecutarlo, no como anteriormente. Por tanto, hablaremos de cómo se realizaron las pruebas en el siguiente capítulo.

CAPÍTULO 5
Pruebas

En esta sección explicamos los experimentos que se han llevado a cabo con la solución propuesta. Se han usado los datos antes mencionados de septiembre de 2018 a julio de 2019 y usando una seed de randomizador para separar entre entrenamiento y test. Definimos además dos conjuntos de usuarios, los que han visto de 1 a 9 vídeos (815 usuarios) y los que han visto de 10 a 150 (1044 usuarios).

Usamos las medidas de precisión y recall. La precisión serían las recomendaciones satisfactorias, es decir, los aciertos, vídeos que ha visto el usuario en el conjunto de test, entre el numero de recomendaciones, mientras que el recall serían los aciertos entre el número de vídeos vistos.

Para hacer experimentos, como queríamos hacer pruebas distintas a las realizadas anteriormente decidimos crear un nuevo archivo py de pruebas con los mismos métodos útiles que tenía el anterior pero con métodos específicos nuevos para el tema que nos incumbía. En concreto, tenemos un método para que se creen las recomendaciones por medio de TF-IDF y otro para que se creen por el nuevo método de Sentence Transformers.

Ambas tienen la misma estructura y se diferencian en que llaman a distinto método, que es el de preprocesar las transcripciones para crear la matriz de similitudes. Se basa en primero crear un objeto de la clase RecommenderSystem, es decir, crear un recomendador por defecto. A partir de él recuperamos la actividad de vídeos deseada, en este caso sería la actividad de septiembre de 2018 hasta julio de 2019, usamos el método *train test split* del paquete sklearn (de scikit) para poder separar los datos en entrenamiento y test.

El conjunto de entrenamiento, que es el 80 %, se le pasa al método para calcular todo lo necesario. En concreto, llama al método *calculate data recommendation user*, que es el que se encarga de llamar al *item profile* o al *item profile bert* dependiendo del test que estemos realizando. Una vez calculado el *item profile*, que es lo más costoso computacionalmente, con él se calcula la matriz de similitud, gracias al método de *cosine similarity*, que se guarda para poder ser utilizada más adelante.

Una vez calculada la matriz de similitudes, con ella y con el 20 % de datos de test, se prueba el recomendador. A cada usuario que exista en el conjunto de test el recomendador le recomendará vídeos. Si no ha habido aciertos, no se añadirá al json de solución, pero si los ha habido, se añadirá al diccionario y lo escribirá en un fichero rec. Este fiche-

ro rec se podrá leer con el process_reco_results para poder extraer las puntuaciones de acierto de las recomendaciones.

Para las métricas usamos el nivel DEBUG del loguru, que se invoca añadiendo al logger la salida sys.stderr y llamando al level "DEBUG". Con este logger, podemos controlar los tiempos de cada uno de los métodos que se llamen durante la ejecución para luego poder hacer comparaciones.

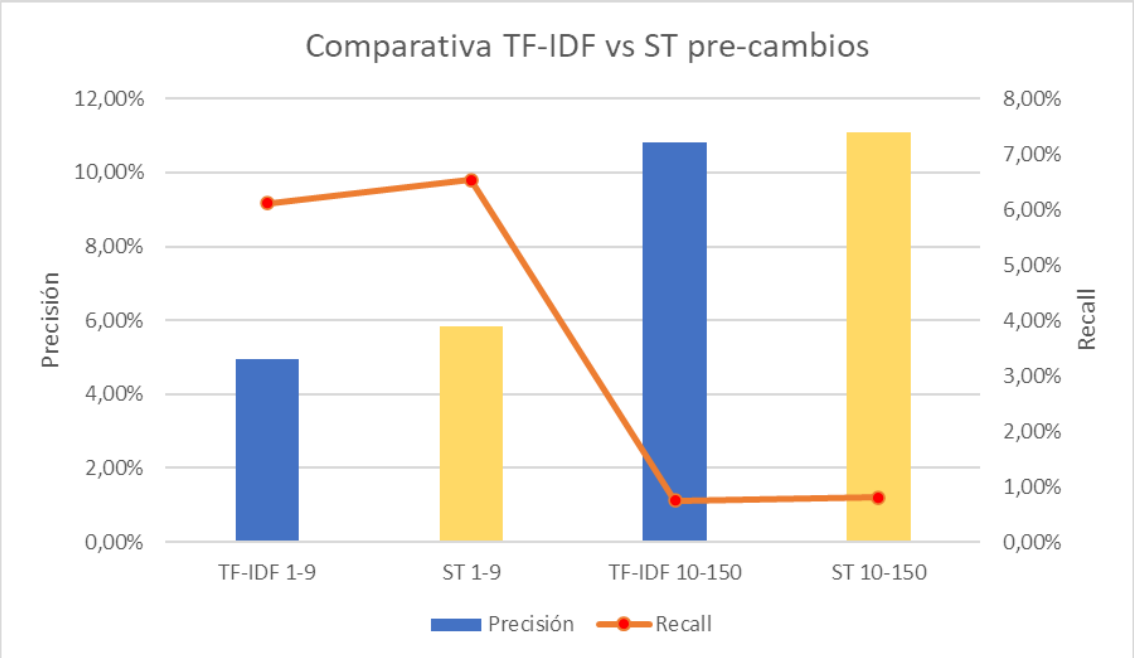
5.1 Análisis de los resultados obtenidos

Las primeras pruebas que se hicieron ya con el Sentence Transformers, se utilizó un tamaño de feature de 32 y el modelo multilingual, además de realizarse antes de cambiar la distribución de los datos. Antes del cambio se tenia de entrenamiento el fichero de actividad de septiembre de 2018 a mayo de 2019, y de test los meses de junio y julio de 2019. Los resultados fueron los siguientes:

Con TF-IDF	Usuarios 1-9	Usuarios 10-150
Precisión	3.3 %	7.2 %
Recall	9.2 %	1.1 %

Con S.Trans	Usuarios 1-9	Usuarios 10-150
Precisión	3.2 %	7.4 %
Recall	8.9 %	1.2 %

Figura 5.1: Comparativa TF-IDF vs ST antes de los cambios



Como podemos observar en las tablas y la gráfica, no nos daba buenos resultados, por lo que cambiamos al método que está actualmente para ver si podían mejorar los mismos. Estos cambios se basaban en subir el número de features, aunque para eso necesitamos cambiar el modelo a uno que permitiese ese número, y es por eso que cambiamos al de hiiamsid [17]. Además de esto, probamos también a cambiar el modo de distribución de los datos para ver si mejoraba algún resultado, probando a juntar todos los datos y a distribuirlos en training y test de forma aleatoria. Los siguientes datos se obtuvieron de esta forma:

TF-IDF		Usuarios 1-9	Usuarios 10-150
Seed 42	Precisión	3.6 %	5.6 %
	Recall	9.4 %	1.6 %
Seed 101	Precisión	3.9 %	6.8 %
	Recall	9.8 %	2.0 %

Los mismos resultados con Sentence Transformers son los siguientes:

S.Transformers		Usuarios 1-9	Usuarios 10-150
Seed 42	Precisión	3.9 %	6.7 %
	Recall	10.6 %	2.0 %
Seed 101	Precisión	3.9 %	6.0 %
	Recall	9.8 %	1.8 %

Figura 5.2: Comparativa TF-IDF vs ST, seed 42

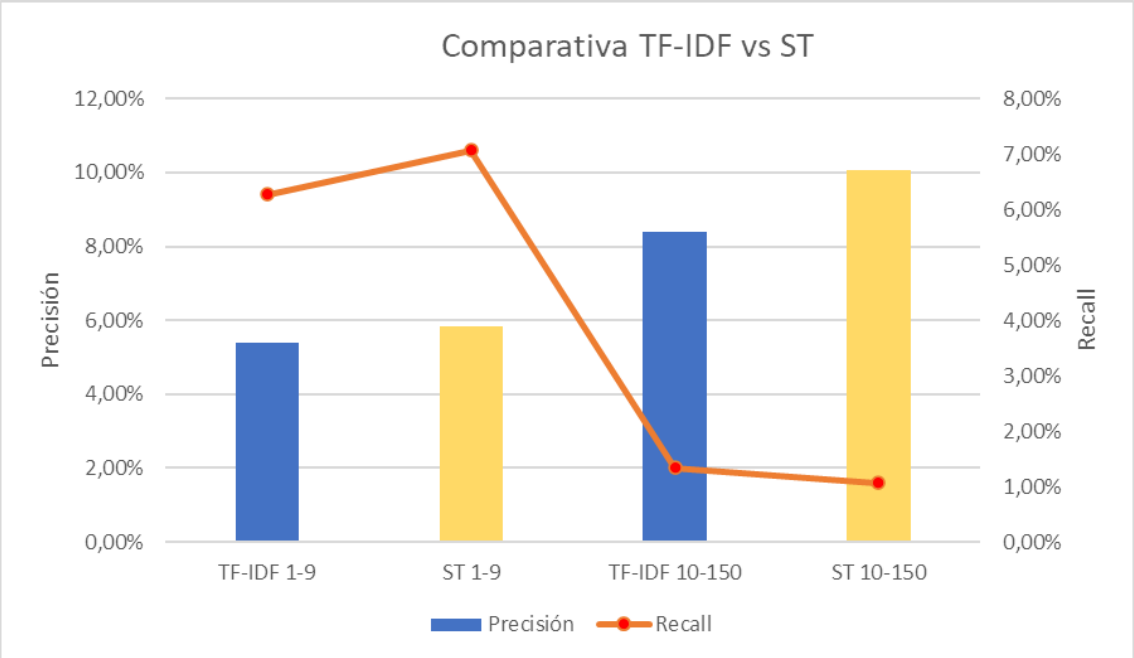
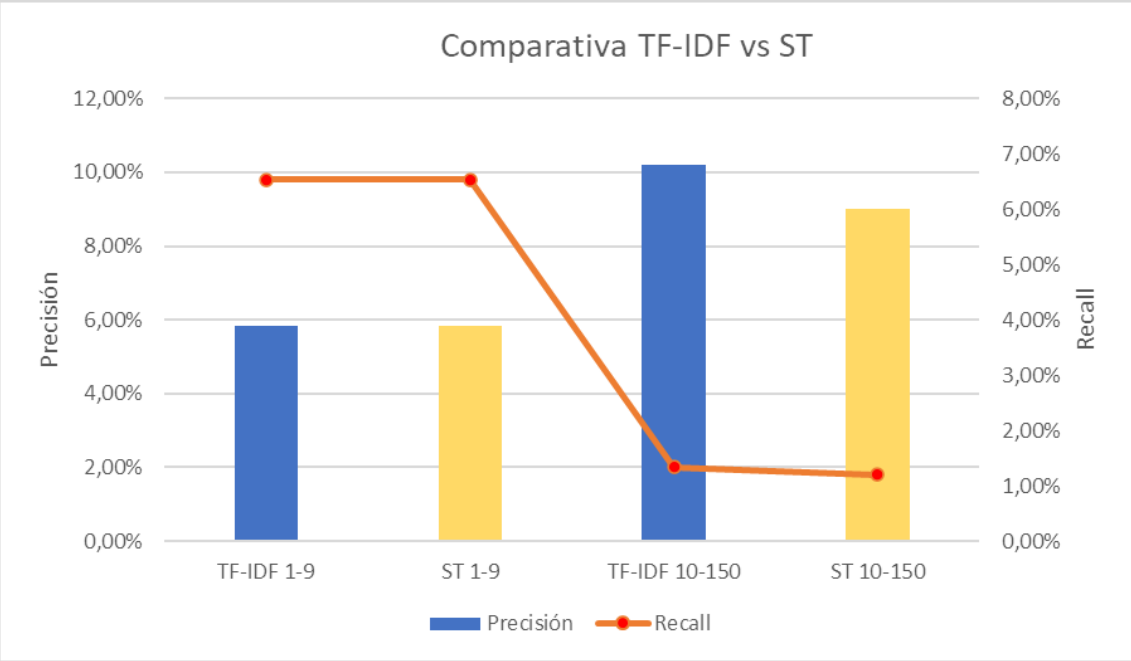


Figura 5.3: Comparativa TF-IDF vs ST, seed 101



Los cambios que se hicieron cambiando el modelo y la forma de organizar los datos aleatoriamente, como se puede apreciar, no fueron cambios a mejor ya que el porcentaje de aciertos bajó notablemente entre ambas pruebas.

Como se ve tanto en las tablas como en las gráficas, se puede observar que no hay tanto cambio en los resultados obtenidos por ambos métodos, TF-IDF y Sentence Transformers, e incluso hay algunos peores como por ejemplo los de la seed 101. Debido a los resultados, podemos observar que no ha habido mejora gracias a la implementación de Sentence Transformers y como hemos visto, en algunas ocasiones puede ser hasta desaconsejable.

No realizamos más tests, primero debido a la falta de tiempo pero, más importante, debido a que los resultados de los primeros tests no fueron satisfactorios ni con el método antiguo de los datos ni con ninguna de las seeds random que probamos, así que lo más probable es que no lo sea en general, aún cambiando algún parámetro.

En cuanto a los tiempos, se puede ver que el método de TF-IDF no tarda casi nada, unos minutos a lo sumo, mientras que el método con el Sentence Transformers puede llevar varias horas de ejecución (el último test fueron 3,07h), ya que tiene que calcular muchas más variables. La parte positiva es que el recálculo de las recomendaciones se realiza una vez al día y no se estaría gastando ese tiempo cada vez que se hace una recomendación, que es casi instantáneo.

Estos resultados no satisfactorios se pueden deber a que el método de Transformers tal cual lo hemos utilizado no tiene un modo de saber qué vídeos son más recomendables que otros ya que no hay un sistema de puntuación en el portal y la única forma que tiene es si se ha visto el vídeo o no, por lo que las recomendaciones no son más acertadas.

CAPÍTULO 6
Conclusiones

Tras la investigación de las nuevas tecnologías, Word2Vec, BERT y Sentence Transformers, la implementación de estas dos últimas y por último la valoración y pruebas de la misma, se pasará a comentar las conclusiones del proyecto.

Durante el desarrollo del trabajo se ha intentado cumplir los objetivos nombrados al principio de la memoria, de los que podemos comentar lo siguiente:

- El primer objetivo que era la investigación de las nuevas tecnologías se ha cumplido bastante bien ya que sí que se ha llegado a investigar las tecnologías que se tenían en mente para el proyecto. Se ha aprendido sobre Word2Vec, BERT y Sentence Transformers y otras tecnologías de vanguardia en el NLP.
- En cuanto a la mejora de la recomendación, no se ha cumplido ya que no ha mejorado en números con respecto a la anterior versión del recomendador. Aún así, se han analizado los problemas ocurridos y las posibles causas, por lo que el trabajo realizado no ha sido en vano.
- El último de los objetivos, que era comparar Word2Vec, TF-IDF y BERT no se ha podido cumplir ya que primeramente, el BERT consumió más tiempo de investigación e implantación del planeado, por lo que no se pudo indagar en el Word2Vec, y además la comparativa entre BERT y TF-IDF no ha sido todo lo satisfactoria que nos hubiera gustado.

A pesar de que la implementación del código fue satisfactoria, los resultados no eran los que esperábamos recopilar. La mejora que supuso implementar el Sentence Transformers al código solo se notó por unas décimas sobre el método de TF-IDF anteriormente usado. Eso sumado a la cantidad de tiempo que llevaba el método para ejecutarse se ha decidido no implementar los cambios en el servidor de la UPV.

Intentando analizar las posibles causas, se ha concluido que el principal problema con el recomendador en comparación a otros recomendadores como podrían ser el de YouTube o el de IMDB, es que no tiene un sistema de graduación de vídeos, y no solo eso, sino que lo más probable es que si lo tuviera no se usara, como ya ocurrió en el pasado. Los alumnos no se quedan después de un vídeo a darle me gusta o no me gusta o darle

puntuación ya que la mayoría de los mismos ven los vídeos para clase y lo cierran. Esto es un gran problema ya que con ello se podrían simplificar mucho las recomendaciones.

Con estas reflexiones, se puede concluir que, pese a que los resultados no han sido buenos, el proyecto ha sido un éxito ya que se ha podido investigar y aprender sobre las nuevas tecnologías de Machine Learning sobre el preprocesamiento de los datos.

6.1 Relación del trabajo desarrollado con los estudios cursados

En la sección se va a relacionar el conocimiento adquirido durante la carrera en sus diferentes asignaturas con el trabajo ahora realizado. Para ello se han seleccionado las asignaturas que más relevancia tienen con el mismo en la siguiente lista:

- APR: Aprendizaje automático sería la asignatura más relacionada con el trabajo en cuestión ya que en la asignatura se enseñan y se practican métodos de Aprendizaje Automático. Además también se introducen las redes neuronales y se trabaja con ellas en el laboratorio.
- SIN: En Sistemas Inteligentes se dan las primeras nociones básicas de Inteligencia Artificial.
- AIN: En Agentes Inteligentes se incrementan los conocimientos enseñados sobre IA.
- GPR: Ayuda para la planificación del tiempo de trabajo, crucial para poder planear de antemano cuanto tiempo va a llevarte cada porción del trabajo.

Además de estas asignaturas, hay muchas anteriores a ellas que son las que dan pie a entender las demás como pueden ser sobretodo las de primer curso, FCO, MAD, PRG, etc., que son las que ponen una base en todos los conocimientos que se han usado para llevar a cabo el trabajo.

6.2 Trabajos futuros

Esta sección se comentan posibles mejoras futuras para el recomendador. Ya que las mejoras que se han planteado no han sido satisfactorias, se podrían plantear futuros objetivos a realizar para mejorar el recomendador. Como el principal problema que tiene el sistema de mediaUPV es que no se basa en puntuación para poder recomendar sino en contenido, es difícil mejorarlo. Aun así se podría intentar tirar por otros métodos de procesamiento del lenguaje natural como los que hemos comentado antes o cambiar por completo el recomendador:

Word2Vec podría ser una opción válida que se podría haber desarrollado en este proyecto pero se gastaron todos los recursos en investigar el funcionamiento y la utilidad de BERT/BETO, por lo que no se pudo investigar en cambio por esta misma. Como se explica en el estado del arte se basa en representar palabras con el contexto y reduciendo la dimensionalidad de esa forma, así que es posible que fuera una alternativa bastante útil y que diera buenos resultados.

Otras opciones sería intentar que los alumnos hicieran graduación de cada vídeo que ven y así poder recomendar vídeos por puntuaciones en lugar de si se ha visto o no, ya que esto no siempre significa que al alumno le haya gustado el vídeo, así que subirían las recomendaciones exitosas, al ser más centradas en los gustos y no en lo visto.

Se podría también crear un perfil de sugerencias y etiquetas para que solo recomiende vídeos con esas características al usuario, seleccionando de la base de datos los vídeos con ciertas etiquetas y luego ejecutando el propio método de recomendación. Por ejemplo que un usuario seleccione que le gusta la Inteligencia Artificial y la Computación y solo le recomiende vídeos con esas características. Las características se podrían obtener a partir del mismo TF-IDF que ya está implementado.

Bibliografía

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Jaume Jordán, Soledad Valero, Carlos Turró, and Vicent Botti. Using a hybrid recommending system for learning videos in flipped classrooms and moocs. *Electronics*, 10(11), 2021.

[3] Universitat Politècnica de València. MediaUPV. https://media.upv.es/.

[4] Joan Albert Silvestre Cerdà, Miguel Angel Del Agua Teba, Gonzalo Vicente Garcés Díaz-Munío, Guillem Gascó Mora, Adrián Giménez Pastor, Adrià Agustí Martínez-Villaronga, Alejandro Manuel Pérez González de Martos, Isaías Sánchez-Cortina, Nicolás Serrano Martínez-Santos, Rachel Nadine Spencer, et al. Translectures. In *IberSPEECH 2012-VII Jornadas en Tecnología del Habla and III Iberian SLTech Workshop*, pages 345–351. IberSPEECH 2012, 2012.

[5] Tom Mitchell, Bruce Buchanan, Gerald DeJong, Thomas Dietterich, Paul Rosenbloom, and Alex Waibel. Machine learning. *Annual review of computer science*, 4(1):417–433, 1990.

[6] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

[7] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[8] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[9] Efficient estimation of word representations in vector space. https://arxiv.org/pdf/1301.3781.pdf.

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[11] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[12] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[13] Torch. https://pytorch.org/docs/stable/torch.html.

[14] Transformers. https://huggingface.co/docs/transformers/index.

[15] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*, 2020.

[16] Github: Beto, spanish version of the bert model. https://github.com/dccuchile/beto.

[17] Hiiamsid model. https://huggingface.co/hiiamsid/sentence_similarity_spanish_es.

APÉNDICE A

Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

De los objetivos antes mencionados, el proyecto está relacionado con los siguientes:

- Educación de Calidad: Pese a que no sea para la principal meta de este ODS, que sería asegurar una educación básica de calidad para todos, sí que podríamos decir que este proyecto podría ayudar a mejorar la calidad de la educación de los estudiantes de la UPV, ya que al recomendar mejores vídeos, se mejora la calidad de vida de los alumnos.

- Industria, innovación e infraestructura: Se trataría de innovación ya que se usan técnicas de recomendación de los últimos años, que son novedosas para este ámbi-

to. Igual que la anterior, no se basa en los mismos puntos ya que en los ODS hablan de un plan más globalizado y esto es a escala pequeña para la Universitat.

Como se ha comentado, no es muy influyente para los ODS ya que los objetivos de las Naciones Unidas son a grande escala y esto no se trata de un TFG que vaya a ayudar a gente más allá de los alumnos de la Universitat. El resto de los puntos no son muy relevantes para el tema del proyecto en cuestión ya que se basan en aspectos que no se relacionan con la innovación en aprendizaje automático ni en los alumnos, ya que es algo que solo servirá para recomendar vídeos en la aplicación web de mediaUPV. No tiene nada que ver el fin de la pobreza o la acción por el clima, entre otras, con el proyecto desarrollado aquí.

APÉNDICE B
Glosario

En este glosario se definirán términos que se han empleado en el trabajo y que pueden ser desconocidos para algunos lectores.

- Aprendizaje Automático/Machine Learning: Rama de la inteligencia artificial que se centra en que las máquinas aprendan solas sin necesidad de programarlo.
- Cosine Similarity: La similaridad coseno es una medida de distancia entre dos secuencias de números, vistas como vectores, calculando el coseno entre ambos vectores.
- Deep Learning: Subrama del Machine Learning que usa algoritmos que funcionan de manera similar a las neuronas del ser humano, es decir, funciona con capas y redes como si imitara las redes de neuronas. Es muy útil en el tema que nos incumbe puesto que se suele usar mucho en el procesamiento del lenguaje natural.
- Inteligencia Artificial: disciplina de las ciencias de la computación centrada en intentar que las máquinas repliquen el pensamiento e inteligencia humanos.
- GitHub: GitHub es una plataforma gratuita de colaboración para desarrollo software en la que se puede almacenar un repositorio de código en la nube para que individuos y grupos puedan trabajar subiendo versiones de una aplicación a la misma.
- Python: Lenguaje de programación de alto nivel que se centra más en que su código sea fácil de leer y de interpretar. Suele usarse mucho en ámbitos como la ciencia de datos, el aprendizaje automático y computación, además de otros ámbitos distintos como el desarrollo web o el de videojuegos.
- Recomendación: acción de aconsejar a alguien sobre lo que puede hacer en su propio beneficio. En este caso concreto sería la acción de que el sistema aconseje a algún usuario sobre que vea un vídeo.
- Red Neuronal: Concepto de la inteligencia artificial que se centra en enseñar a las maquinas a procesar información como si de una red de neuronas biológica se tratase. Se ayuda de los algoritmos de deep learning para llevar a cabo su función.

- Tensorflow: Plataforma/biblioteca de código abierto para el machine learning, desarrollada por Google y que cuenta con un sistema amplio de herramientas y recursos para ayudarte con tus proyectos de aprendizaje automático.
- UPV: Universitat Politècnica de València, es una universidad pública española, concretamente en Valencia, como dice su nombre. Tiene en su plan de estudios 39 grados y 41 centros de investigación.