Personalización de rutas en transporte público usando opiniones de usuarios: Aplicación al metro de Madrid
Personalization of public transport routes using user feedback: Application to the Madrid subway system

Trabajo de Fin de Grado
Curso 2024–2025

Autor

José María Martín Dasilva (GII) - 9.5
Daniel Pérez García (GII) - 9,5
Francisco Prieto Gallego (GII) - 9.5
Ibon Malles Altolaguirre (GII) - 9.5

Director

Manuel Méndez Hurtado
Manuel Núñez García

Grado en Ingeniería Informática
Universidad Complutense de Madrid

Personalización de rutas en transporte público usando opiniones de usuarios: Aplicación al metro de Madrid
Personalization of public transport routes using user feedback: Application to the Madrid subway system

Final Degree Project in Computer Engineering and Software Engineering

Autor

José María Martín Dasilva (GII) - 9.5
Daniel Pérez García (GII) - 9,5
Francisco Prieto Gallego (GII) - 9.5
Ibon Malles Altolaguirre (GII) - 9.5

Director

Manuel Méndez Hurtado
Manuel Núñez García

Call: *June 2025*

Grado en Ingeniería Informática
Universidad Complutense de Madrid

20 de Mayo de 2025

Dedicatoria

A los compañeros de la UCM y a nuestras familias por aguantarnos en este tiempo

Agradecimientos

A nosotros. A Ibon, por orientarnos y ayudarnos. A Fran, por su implicación en todo el proyecto. A José María, por ayudar y colaborar sin descanso y a Daniel por su labor e implicación.

Resumen

Personalización de rutas en transporte público usando opiniones de usuarios: Aplicación al metro de Madrid

Este proyecto tiene como objetivo desarrollar una aplicación web que personalice las rutas de transporte público en el Metro de Madrid utilizando opiniones y valoraciones de usuarios. Se usarán técnicas de Procesamiento de Lenguaje Natural para analizar y clasificar las reseñas de los usuarios para optimizar el trayecto según diversas preferencias y criterios específicos. Ejemplos de estos criterios son solicitar que los trasbordos se realicen en estaciones, preferir líneas con valoraciones positivas y evitar aquellas con valoraciones negativas, minimizar el tiempo en estaciones con alta percepción de inseguridad o priorizar rutas que pasen por estaciones destacadas por su accesibilidad para personas con movilidad reducida.

Palabras clave

Transporte público, personalización de rutas, web scrapping, NLP, aumento de datos, API, encapsulación, evaluación con usuarios,

Repositorio GitHub

https://github.com/IBTARK/TFG.git

Abstract

Personalization of public transport routes using user feedback: Application to the Madrid subway system

This project aims to develop a web application that personalizes public transport routes in the Madrid Metro using user opinions and ratings. Natural Language Processing techniques will be used to analyze and classify user reviews in order to optimize the route according to different preferences and specific criteria. Examples of these criteria are requesting that transfers be made at stations, preferring lines with positive ratings and avoiding those with negative ratings, minimizing time in stations with high perception of insecurity or prioritizing routes that pass through stations noted for their accessibility for people with reduced mobility.

Keywords

Public transportation, route customization, web scrapping, NLP, data augmentation, API.

GitHub Repository

https://github.com/IBTARK/TFG.git

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Todo parece imposible hasta que lo haces"
— Nelson Mandela

Este proyecto tiene como finalidad el desarrollo de una aplicación web orientada a la personalización de rutas de transporte público en la red del Metro de Madrid, basándose en las valoraciones y opiniones proporcionadas por los propios usuarios. Para ello, se aplicarán técnicas de Procesamiento de Lenguaje Natural (NLP por sus siglas en inglés) con el objetivo de analizar y clasificar automáticamente las reseñas, permitiendo así adaptar las rutas a diferentes preferencias y criterios definidos por el usuario.

1.1. Estructura de la memoria

La memoria se va a estructurar en cinco partes para facilitar la lectura:

- **Introducción**: empezará con una breve introducción donde se mostrará tanto la motivación para llevar a cabo el proyecto como los distintos objetivos para su correcta implementación. Además, se explicará la metodología de trabajo que el grupo va a llevar a cabo.
- **Estado del Arte**: se realizará un estudio de las tecnologías y aplicaciones que podría tener nuestro proyecto, analizando trabajos y estudios previos e indicando las aplicaciones que existen en el mercado actual que podrían competir con la nuestra.
- **Descripción del trabajo**: se indicará cada uno de los pasos que se realizarán para la implementación del proyecto final, tanto de la parte de obtención de datos como del desarrollo del *frontend* y del *backend*.
- **Contribuciones Personales**: se describirán las actividades realizadas por cada miembro del grupo para llevar a cabo este proyecto.

- **Conclusiones y Trabajo futuro**: se resumen los resultados obtenidos y se indicarán las limitaciones que puedan haber. También se incluirán objetivos que no se han podido alcanzar o que se puedan mejorar en un futuro.

1.2. Motivación

El transporte público juega un papel clave en la vida cotidiana de millones de personas, especialmente en grandes ciudades como Madrid. Las personas que usan el metro con frecuencia, han detectado que muchas de las aplicaciones existentes para planificar rutas carecen de opciones de personalización y no siempre ofrecen trayectos que se adapten a las preferencias individuales. En concreto, factores como la limpieza de las estaciones, la accesibilidad, la seguridad o la afluencia de personas no suelen tenerse en cuenta en los algoritmos de recomendación. Ante esta situación, se ha considerado que existe una oportunidad de mejora mediante el uso de reseñas y valoraciones realizadas por los propios usuarios del metro. Este proyecto propone el desarrollo de una aplicación, que haciendo uso de las reseñas introducidas por los usuarios de este transporte, permita calcular rutas personalizadas entre estaciones del metro de Madrid, integrando los filtros que cada usuario considere más relevantes.

1.3. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es el desarrollo de una aplicación web que recomiende rutas en el metro de Madrid en función de una serie de filtros seleccionados por el usuario. Para ello, se extraerán las reseñas de todas las estaciones de metro de madrid que se encuentren en plataformas como Google Maps, se diseñará un sistema capaz de procesar y clasificar automáticamente las opiniones, extrayendo información útil sobre aspectos como la limpieza, seguridad o accesibilidad de las estaciones. A partir de esta información, la aplicación ofrecerá rutas personalizadas que se ajusten a los filtros seleccionados por cada usuario. Además, se busca implementar una interfaz intuitiva y funcional que facilite el uso de la herramienta por parte del público general.

1.4. Metodología de trabajo

El proyecto se desarrollará en varias fases. En primer lugar, se recopilarán los datos necesarios sobre las estaciones del metro de Madrid, incluyendo reseñas de usuarios procedentes de fuentes públicas. Posteriormente, se construirá una base de datos para almacenar esta información de forma estructurada. A continuación, se aplicarán técnicas de NLP para clasificar las reseñas y extraer sus características. Paralelamente, el desarrollo se dividirá en dos bloques: el *backend*, encargado de la lógica de procesamiento y recomendación, y el *frontend*, centrado en el diseño de la interfaz de usuario. El equipo se organizará en dos subgrupos que trabajarán

de forma coordinada para garantizar la integración de todos los componentes del sistema.

Capítulo 2

Estado del Arte

En esta sección se van a revisar los avances tecnológicos relacionados con el cálculo de rutas personalizadas, el procesamiento de datos de usuarios y su aplicación en sistemas de transporte público como el Metro de Madrid.

2.1. Introducción

Con el avance de la tecnología, el uso de navegación y planificación de rutas se ha convertido en una parte fundamental para la movilidad tanto en vehículos privados como en el transporte público. Esto se debe a que los usuarios confían más en aplicaciones o GPSs que le ayuden a encontrar el trayecto más rápido, cómodo y eficiente. Este aumento en la demanda ha impulsado el desarrollo de algoritmos de cálculo de rutas más sofisticados, que van más allá de los enfoques tradicionales basados únicamente en tiempo o distancia. En la actualidad, muchos de estos algoritmos incorporan factores subjetivos, integrando opiniones y valoraciones de los propios usuarios con el fin de generar rutas personalizadas que se ajusten mejor a sus preferencias individuales.

2.2. Transfondo

El cálculo de rutas es un tema bastante estudiado en el ámbito de la informática. Se basa en calcular el camino mínimo entre dos puntos mediante el tiempo o la distancia. Generalmente, está basado en grafos, donde cada nodo representa un punto (ciudad, estación, etc) y las aristas la conexión entre ambos ponderadas. Los algoritmos más utilizados para resolver esta clase de problemas son Dijkstra, A* y sus variantes, centrados siempre en la minimización del coste basado en tiempo o distancia. Sin embargo, se están modificando estos algoritmos con el objetivo de tener en cuenta las opiniones que los propios usuarios han reportado a través de reseñas en aplicaciones o redes sociales como la limpieza, la seguridad o la accesibilidad.

Con el objetivo de incorporar estos nuevos factores en el proceso de generación de

rutas, se han comenzado a aplicar técnicas de NLP. Estas técnicas permiten extraer información relevante a partir de los comentarios y reseñas de los usuarios, lo que posibilita la creación de rutas optimizadas que consideren no solo criterios objetivos, sino también percepciones subjetivas expresadas en lenguaje natural.

2.3. Revisión de la Literatura

Para la obtención de las rutas personalizadas se usan las mezclas de distintas tecnologías previamente descritas como algoritmos de obtención de rutas, grafos y NLP.

El algoritmo de Dijkstra para la obtención del camino de coste mínimo y A* que acelera la búsqueda usando heurísticas admisibles son la base de casi todas las soluciones de ruta en grafos. Hay muchos más algoritmos como el de Bellman-Ford que permite trabajar con aristas de coste negativo (al contrario que Dijkstra), Floyd-Warshall que calcula la ruta más corta entre todos los pares de nodos en un grafo o el de K de Yen que busca las K rutas más cortas sin bucles desde un nodo fuente a un nodo destino, ofreciendo opciones alternativas.

El NLP es muy importante para obtener la información más relevante de los comentarios que hacen los usuarios que es la principal fuente de obtención de información para poder calcular las rutas.

Recientemente se han creado algoritmos basados en Dijkstra para crear rutas personalizadas, por ejemplo, Samuel Andrés Areiza (2019) et al lo modifican para encontrar la ruta más segura y óptima para poder así evitar los abusos sexuales y disminuir la inseguridad callejera. Alican Bozyiğit (2017) et al muestran como lo modifican para penalizar transferencias y tramos de caminata excesiva, de modo que las rutas propuestas sean más cercanas a lo que los viajeros consideran “ideales”.

2.4. Desafíos y limitaciones

Los datos juegan un papel muy importante. Uno de los mayores desafíos a la hora de implementar estos algoritmos basados en personalización de rutas, es la cantidad y calidad de los mismos. Al obtenerlos de reseñas de distintos ámbitos como foros, redes sociales o Google Maps pueden ser escasos y contener información irrelevante o con ruido. Además, el volumen de las reseñas entre las distintas reseñas puede variar mucho, resultando en una gran diferencia entre el número de reseñas entre unas estaciones y otras. Otro de las limitaciones que te puedes encontrar es el procesamiento de las mismas mediante el uso de técnicas de NLP. Esto se debe a que esta clase de métodos estan poco desarrolladas en Español.

2.5. Avances recientes

La personalización de rutas de transportes públicos ha evolucionado con rapidez en los últimos años gracias sobre todo a la LLM-as-a-service, estándares GTDS y datos de aforo en tiempo real. Cómo ejemplo más relevante y actual de avances de IA tenemos Google Maps + IA Gemini que trata de un chat que google maps ha integrado que resume reseñas, contesta diferentes preguntas y reordena rutas según diferentes criterios. Por parte de Apple Maps también ha habido un reciente avance en el cuál puedes introducir rutas a medida y sin conexión generadas a partir del historial y preferencias del usuario. Como ejemplo de Deep Learning sobre grafos, tenemos PNMLR que dado un usuario, una hora y una característica predice la ruta más probable.

2.6. Casos Reales

Tras realizar un profundo estudio acerca de las diferentes aplicaciones que hay en el mercado que implementen algoritmos para la recomendación de rutas hemos obtenido las siguientes:

- **App oficial Metro de Madrid**: es la aplicación oficial del metro de Madrid. Esta desarrollada y mantenida por Metro de Madrid S.A. Entre sus funcionalidades destaca la obtención de rutas entre sus distintas estaciones de metro con una visualización de un mapa interactivo para ver el trayecto que el usuario va a realizar o la búsqueda de las estaciones. Su planificación de rutas está controlada a través de geolocalización GPS para indicar la estación más cercana al usuario. Además, te proporciona la ruta óptima en función de criterios como tiempo de viaje, número de transbordos o distancia con la capacidad de proporcionar rutas alternativas. Tiene también la posibilidad de dar información acerca del estado de las líneas en tiempo real para poder consultar los horarios, las posibles incidencias que puedan ocurrir y que estaciones tienen acceso para personas con movilidad reducida. Los datos los obtiene a través de Open Data feeds de Metro de Madrid para obtener información en tiempo real sobre el estado de las líneas y estaciones.

- **MLO (Metro ligero oeste)**: es una aplicación sobre las estaciones de metro ligero ML2 y ML3. La empresa encargada de su desarrollo y mantenimiento es Metro Ligero Oeste S.A. Es capaz de crearte trayectos en tiempo real entre dos paradas del metro ligero y la incorporación de conexiones con metro y cercanías. Además, incluye un mapa interactivo de las líneas del metro ligero oeste con geolocalización del usuario y búsqueda de paradas. También incluye un reporte de incidencias y noticias sobre el metro. Consume los feeds de Open Data del Consorcio Regional de Transportes de Madrid (CRTM) para obtener horarios y estado de servicio en tiempo real. Incluye además información para personas con movilidad reducida. Proporciiona información en tiempo real y

una estimación de tiempo para llegar al destino. Utiliza feeds GTFS proporcionados por más de 100 autoridades de transporte para la red estática (líneas, paradas, horarios) y GTFS-Realtime para posiciones de vehículos y alertas .

- **Citymapper**: es una herramienta global de planificación multimodal de viajes que integra datos de transporte público, transporte activo (caminata, bicicleta, patinete) y servicios de movilidad compartida (taxi). Está desarrollada por Citymapper Limited, que también es la empresa encargada de su manteniiento. Entre sus funcionalidades se encuentra la creación de rutas a partir de tres filtros distintos a elegir por el usuario: más rápida, más seguras y menos transbordos.

- **Moovit**: es una aplicación de personalización de rutas desarrollada por Moovit Inc. En ella se pueden realizar rutas en diferentes medios de transporte con opción a elegir. El usuario puede visualizar en tiempo real el tiempo que va a tardar el medio en llegar a su punto de origen para iniciar la ruta, además de poder hacer un seguimiento de la ruta en todo momento. Otras funcionalidades a destacar serían la de poder ver información de cualquier parada en ese momento y la de poder consultar información de todas las rutas como las paradas que contienen, información de esas paradas y el horario que tienen. Además, la aplicación manda alertas para avisar si ha habido alguna incidencia o dar información sobre posibles cambios en el itinerario de las rutas.

- **Google Maps**: es una de las aplicaciones de geolocalización más usadas en todo el mundo. Pertenece a Alphabet Inc., empresa matriz de Google LLC. Entre sus funcionalidades destacan los mapas desplazables, las imágenes por satélite y la vista a pie de calle utilizando la tecnología de Google Street View. Además permite ver sitios de interés como restaurantes, gasolineras, entre otras. Además, gracias a Google Local se pueden encontrar resultados restringidos a una zona.

- **Madrid Metro Bus Cercanías (Madrid MBC)**: es una aplicación de transporte público desarrollada por Bidea Digital Services. Esta app integra en una sola plataforma información en tiempo real de los principales medios de transporte de la Comunidad de Madrid: Metro de Madrid, EMT (autobuses urbanos), autobuses interurbanos, Cercanías RENFE y BiciMAD. La aplicación permite al usuario consultar los tiempos de llegada por estación y andén, calcular trayectos entre dos estaciones y visualizar mapas esquemáticos, geográficos y turísticos del Metro de Madrid. Madrid MBC está disponible para dispositivos Android e iOS y se actualiza continuamente para incorporar las últimas novedades y variaciones en los servicios de transporte.

- **Apple Maps**: Es una aplicación de navegación desarrollado por Apple Inc. Ayuda a los usuarios a crear itinerarios en diferentes medios de traslados, como en coche, bicicleta, a pie o en transporte público, y brinda múltiples opciones para llegar a la localización. La aplicación calcula estimaciones de tiempo de

llegada en tiempo real, considerando el tráfico y el estado del transporte público, y permite seguir la ruta de forma detallada con pasos instructivos. Entre sus funcionalidades destaca la vista tridimensional de ciudades, la integración con Siri para navegación por voz, y la posibilidad de ver información en tiempo real sobre las líneas de metro, autobuses, trenes y sus horas de servicio y posibles interrupciones. También, Apple Maps otorga alertas para notificar al usuario sobre cambios en las rutas, accidentes, o mal clima durante el viaje.

Capítulo 3

Descripción del Trabajo

En este capítulo se detallan todas las fases que componen el desarrollo práctico del proyecto, desde la obtención y procesamiento de los datos, hasta la construcción de la base de datos, el diseño del clasificador de reseñas, la creación de rutas y la implementación de la aplicación web. Cada sección recoge los pasos seguidos, las herramientas utilizadas, las decisiones tomadas y los problemas encontrados, así como las soluciones adoptadas para avanzar en cada etapa del trabajo.

3.1. Obtención de datos

El objetivo de esta primera fase fue extraer el máximo número de reseñas posibles de las doscientas setenta y seis estaciones de metro de Madrid.

3.1.1. Elección de la herramienta

El primer paso consistió en escoger la plataforma de la cual se iban a extraer los datos. De todas las opciones posibles (Google Maps, Apple Maps, Yelp, Trip Advisor, Booking..), las únicas con reseñas de estaciones de metro de Madrid eran Google Maps y Yelp. Pero, esta última solo dispone de un número muy limitado de reseñas de las estaciones más emblemáticas. En consecuencia, la extracción de reseñas se limitó únicamente a Google Maps.

Para el proceso de recolección de las reseñas, la primera opción fue hacer uso de la técnica conocida como *Web Scraping*, la cual consiste en usar software específico para extraer información de páginas web.

Inicialmente se propuso utilizar la extensión de Chrome conocida como *Instant Data Scraper*, pero se descartó debido a la imposibilidad de automatizar el proceso. Al activar esta extensión y realizar la búsqueda "estaciones metro Madrid" en *Google Maps* se obtenía la lista de estaciones de Madrid. Sin embargo, no era posible acceder a las reseñas de cada estación. Para conseguirlo sería necesario *scrapear* una a una las correspondientes páginas de cada estación.

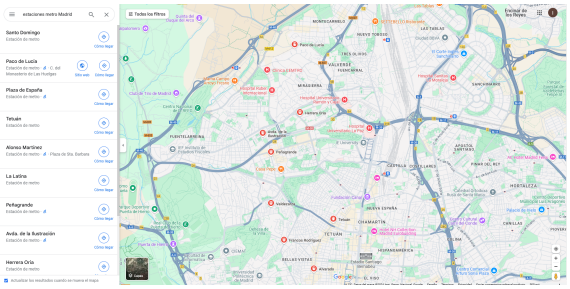

Figura 3.1: Lista de estaciones de Madrid

Para acceder a las reseñas de forma más rápida y efectiva se decidió hacer uso de la API `SerpApi` que permite realizar búsquedas en motores de búsqueda de forma automatizada. Este método de obtención de los datos fue rápidamente descartado debido a la limitación de peticiones disponibles. Al crearse una cuenta de SerpApi se dispone de cien búsquedas de forma gratuita, las cuales se gastan de forma inmediata. La extracción de reseñas de una estación con aproximadamente quinientas reseñas hace uso de todas las búsquedas gratis. Esto se debe a que cada llamada permite acceder a una página de reseñas de una estación concreta, la cual contiene entre cinco y diez reseñas. Consecuentemente, para extraer las quinientas reseñas se hace uso de las cien búsquedas. Como ninguno de los planes de pago de SerpApi eran económicamente asequibles (el más barato son setenta y cinco euros al mes por cinco mil búsquedas y el más caro son ciento seis mil euros por cincuenta y cuatro millones de búsquedas), el grupo se vio obligado a utilizar otro método de extracción de datos.

Como alternativa se programó un bot utilizando Python, más concretamente la librería de Selenium. Ésta es una herramienta que proporciona una interfaz para interactuar y controlar los navegadores web de forma automatizada. El principal componente es un *WebDriver* (una interfaz de control remoto), que actúa como puente entre el script escrito y el navegador.

Desafortunadamente, utilizar un bot para *scrapear* Google Maps infringe los términos impuestos por la compañía. Consecuentemente, se tuvo que descartar este enfoque.

Se revisaron otras alternativas, entre ellas Data Miner, Web Scraper y Agently, pero finalmente el grupo se decantó por volver a la herramienta inicial (Instant Data Scraper).

3.1.2. Instant Data Scraper

Instant Data Scraper es una herramienta automatizada de extracción de datos de páginas web. No requiere de scripts específicos para las páginas web ya que utiliza Inteligencia Artificial (IA) guiada por heurísticos para el análisis del código HTML. Además de la extracción de datos, detecta cuando los datos dinámicos se han cargado, permite seleccionar el tiempo mínimo y máximo de espera para la carga de datos al *scrollear* y automatiza el paso de páginas mediante el botón “next” o

mediante links. Una vez obtenidos los datos, los muestra en una tabla con tantas filas como datos obtenidos y con un número de columnas determinado que te permite modificar. Finalmente, los datos pueden ser exportados a Excel o a un fichero CSV.

Haciendo uso de esta herramienta, cada miembro del grupo hizo *scrapping* de sesenta y nueve estaciones, obteniendo entre todos las reseñas de todas las estaciones de metro de Madrid.

Para *scrapear* las reseñas de cada estación de metro se siguió el siguiente proceso: primero se abrió el navegador Chrome y se buscó la página de Google Maps correspondiente a la estación de metro. A continuación, se accedió a la sección de reseñas y se desplegaron todas ellas (las reseñas más largas no se ven al completo, por lo que hay que hacer clic en el "más"). Esto último forzó a *scrollear* hasta el final de la página. Seguidamente, se ejecutó la extensión de Chrome Instant Data Scraper, la cual, como se puede ver en la Figura 3.2 abre una ventana con los datos obtenidos en forma de una tabla. La primera vez que se ejecutó esta extensión los datos no eran los relevantes (las reseñas). Consecuentemente, se pulsó el botón "Try another table" hasta que las reseñas se mostraron en la tabla. Las siguientes ejecuciones de la extensión mostraban la misma tabla a la primera. Dado que no solo se mostraban las reseñas, se eliminaron las columnas innecesarias (por ejemplo, las correspondientes a las fotos adjuntadas a los comentario), obteniendo finalmente una tabla con las columnas correspondientes al autor de la reseña, información del autor, antigüedad de la reseña y la reseña.

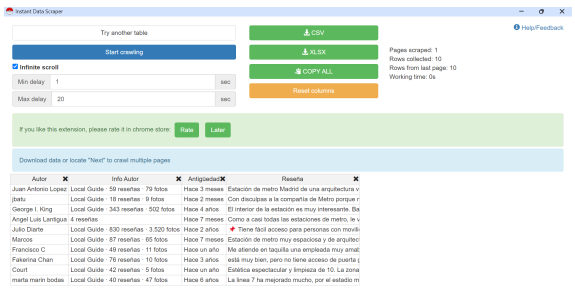

Figura 3.2: Resultado Instant Data Scraper

Finalmente, los datos *scrapeados* se exportaron a ficheros CSV. De esta forma, se obtuvieron doscientos setenta y seis ficheros, uno para cada estación.

3.1.3. Problemas de Instant Data Scraper

El primer problema se identificó al abrir los ficheros CSV: las reseñas que contenían saltos de línea también los mantenían dentro del CSV. Como consecuencia, los ficheros estaban corrompidos, ya que no todas las filas tenían el mismo número de elementos. Para solucionar este contratiempo, se programó un script en Python que detectaba las líneas erróneas y generaba un *.txt* indicando la posición de los errores. Haciendo uso de las indicaciones, cada miembro del grupo revisó los ficheros que le correspondían eliminando los saltos de línea innecesarios.

El segundo problema se identificó en la siguiente etapa del proyecto. Al procesar los datos se identificaron lineas que contenían un distinto número de elementos que las del resto de filas del fichero. La solución fue simplemente añadir dobles comillas en aquellas filas que contenían un distinto número de elementos.

3.2. Tratamiento de datos

Una vez descargados los ficheros CSV, se solucionó el primer problema generado por Instant Data Scraper. A continuación, se programó un script en Python para extraer a otros CSVs únicamente las reseñas (se prescindió en estos ficheros del nombre del autor, información del autor y antigüedad de la reseña). En este momento, el segundo problema (anteriormente mencionado) se identificó. El resultado de ejecutar este código fue un fichero CSV por cada estación con únicamente las reseñas que correspondían. Además, se eliminaron las reseñas vacías.

Por último, se juntaron todas las reseñas de todas las estaciones en un único fichero. De este modo, se obtuvieron un total de 16835.

3.3. Clasificador de reseñas

Al leer un subconjunto las reseñas, el grupo se dio cuenta de que no todas eran útiles. Por ejemplo, una gran cantidad de los datos obtenidos para las estaciones de metro del Aeropuerto Adolfo Suárez Madrid-Barajas no se referían a la estación de metro sino al Aeropuerto. Este mismo problema se daba en bastantes estaciones de Madrid. Además, muchos usuarios de Google Maps adjuntan reseñas sin sentido para conseguir puntos que les permiten recibir recompensas y acceso anticipado a nuevas funciones. Esto último aumenta el número de reseñas inválidas. Para hacer frente a este problema, se decidió generar un clasificador para determinar las reseñas útiles.

3.3.1. Preparación del dataset

Se generó un código en Python que de forma aleatoria, con una probabilidad del veinte por ciento, seleccionase una reseña y la mostrase por pantalla para ser etiquetada. Cada reseña se etiquetó con “v” (válida) o “n” (no válida). Finalmente, se obtuvo un CSV de tres mil doscientos treinta y seis filas, cada una con tres elementos: el fichero origen, la reseña y su etiqueta.

3.3.2. Aumento de datos

El número de datos disponibles era muy reducido para generar cualquier tipo de clasificador con una precisión aceptable. Consecuentemente, se decidió aplicar la técnica conocida como aumento de datos. Éste proceso es clave para mejorar el

rendimiento y la generalización de los modelos, especialmente en casos como este, donde el conjunto de datos es limitado. Consiste en generar nuevas muestras a partir de los datos existentes aplicando diversas transformaciones. El aumento de datos es muy popular en aplicaciones de visión computacional, pero es igualmente poderoso para modelos de NLP.

A continuación se van a presentar las distintas técnicas que se aplicaron.

3.3.2.1. Retrotraducción

La retrotraducción consiste en traducir las oraciones a otro idioma y de vuelta al original para obtener variaciones.

El primer paso para aplicar la retrotraducción fue buscar una forma para acceder a las APIs de los traductores. Primero se intentó usar la librería `Translators` de Python, pero debido a la sintaxis confusa y a los fallos de conexión se descartó. La siguiente opción, `GoogleTrans`, también se descartó debido a fallos de conexión que no permitían completar todas las traducciones necesarias. Por último, la librería que finalmente se utilizó fue `GoogleTrans`. Ésta permite acceder a varias APIs, pero las únicas gratuitas para la cantidad de datos a traducir eran las de Google y MyMemory. Sin embargo, la segunda dio errores de conexión continuamente, por lo que finalmente solo se utilizó el traductor de Google.

La clave para el éxito al aplicar la retrotraducción reside en la elección de los idiomas a los que se traduce. Idiomas más cercanos o similares al original (como el español y el inglés) suelen introducir menos cambios estructurales o de significado. Por otro lado, idiomas con estructuras gramaticales o vocabulario muy distintos (como el español y el japonés), tienden a producir frases finales con más variabilidad en la construcción. Además, usar múltiples idiomas intermedios puede maximizar la variabilidad. Teniendo esto en cuenta, para maximizar el potencial de esta técnica, se generaron tres conjuntos de datos. El primero fue el más simple: se tradujeron los datos originales al inglés y de vuelta al castellano. En el segundo, los datos fueron traducidos al japonés y de vuelta al castellano. Finalmente, el último conjunto de datos se generó traduciendo los datos del español al francés, del francés al japonés, del japonés al ruso y por último del ruso al español. Llegados a este punto, por cada frase del conjunto de datos original se habían generado tres más. Sin embargo, seleccionar las tres representaciones podría sesgar el modelo. En general, las traducciones son muy similares entre sí, lo cual podría generar que el modelo aprendiese patrones específicos en lugar de generalizar. Para evitar este problema, se decidió seleccionar solo una de las tres traducciones.

Para seleccionar las frases adecuadas, era necesario poder compararlas entre sí. Esto requería una representación de los textos que pudiera ser procesada por un modelo computacional. Existen dos enfoques principales para representar textos mediante *embeddings* (representaciones numéricas que convierten palabras o frases en vectores en un espacio de menor dimensión): los *embeddings* a nivel de palabra y los *embeddings* a nivel de documento. En el primer caso, un texto se representa como un conjunto de *embeddings* de palabras. Es decir, la representación es una secuencia

de vectores individuales, cada uno correspondiente a una palabra. En el segundo caso, el texto entero se transforma en un único vector que captura la información global del contenido. Dado que el objetivo era comparar textos completos, se optó por el enfoque a nivel de documento. Para obtener estas representaciones de los textos se usó el modelo `MiniLM-L6-v2`, que es una versión más ligera del modelo `SBERT` (Sentence Bidirectional Encoder Representations from Transformers), entrenada para ser más rápida y eficiente en la generación de *embeddings*. `SBERT` es un versión de `BERT` diseñada específicamente para crear representaciones de oraciones (*embeddings* de oraciones) de forma más eficiente. Todos estos modelos se entrenaron usando un conjunto de datos en varios idiomas, entre ellos el español, de modo que no hubo problema al introducir textos en este idioma. Se usó `MiniLM` para mapear cada texto a un espacio de trescientas ochenta y cuatro dimensiones, obteniendo de esta forma una representación más fácil de manejar.

La similitud semántica, una de las dos principales formas de comparar textos, genera una medida cuantitativa del parecido del significado de dos textos. Se implementó éste método de similitud obteniendo los *embbedings* de pares de oraciones del modelo `MiniLM-L6-v2` y aplicando la similitud del coseno entre los vectores devueltos. En NLP, casi siempre se trabaja con vectores que no tienen valores negativos en sus dimensiones, por lo que los valores de similitud del coseno suelen estar en el rango cero (los textos no se parecen en significado) y uno (los textos son idénticos semánticamente).

$$\text{Similitud del Coseno} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Figura 3.3: Fórmula de la Similitud del Coseno

La segunda métrica que se utilizó fue la similitud léxica, que mide el grado de coincidencia de palabras o términos entre dos textos, sin tener en cuenta el significado subyacente. Existen diversas técnicas para calcular la similitud léxica (como la similitud del coseno basada en frecuencia de palabras o el coeficiente de Dice), pero dado que la frecuencia de las palabras no era relevante para este estudio, se optó por el coeficiente de Jaccard. Este coeficiente, definido en Teoría de Conjuntos y ampliamente utilizado en NLP, indica la similitud entre dos conjuntos y se define como la intersección de palabras entre dos textos, dividida por su unión:

$$JS(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Figura 3.4: Fórmula del Coeficiente de Jaccard

Para dar más valor a las palabras más representativas de los textos, se eliminaron las "palabras vacías", que son aquellas de aparición frecuente y sin contenido significativo, y se eliminaron los signos de puntuación y acentuación.

Disponiendo de formas de representar y comparar textos, se computaron las similitudes semánticas y léxicas de cada texto del conjunto de datos original con respecto a las retrotraducciones generadas. Finalmente, para cada texto original se seleccionó aquella traducción que maximizaba la siguiente fórmula:

$$SimilitudSemantica - SimilitudLexica$$

Consecuentemente, se duplicó el conjunto de datos únicamente aplicando la técnica de retrotraducción.

3.3.2.2. Reemplazo por sinónimos

El reemplazo por sinónimo se basa en elegir aleatoriamente n palabras de un texto y sustituirlas por sinónimos.

Texto original:

texto = La estación está sucia porque no la limpian.

Se convierte en:

texto = La estación está mugrienta porque no la limpian.

Figura 3.5: Reemplazo por sinónimos

Para implementar esta técnica, se investigaron diversas librerías de Python. Desafortunadamente, el número de opciones disponibles para obtener sinónimos en español es limitado, y la mejor alternativa encontrada fue `WordNet`, accesible a través del módulo `nltk.corpus`. Aunque esta librería está diseñada principalmente para inglés, descargando el módulo `omw-1.4`, es posible acceder a sinónimos en otros idiomas (entre ellos el español). Aún así, hay bastantes limitaciones, siendo la más notable la obtención de sinónimos de palabras en plural. A pesar de esta restricción significativa, `WordNet` fue la única opción viable para un corpus en castellano.

Una vez obtenido el acceso a los sinónimos en español, el siguiente paso fue seleccionar aleatoriamente un subconjunto de palabras de cada reseña para sustituirlas por sus sinónimos. La cantidad de palabras a sustituir se determinó como un porcentaje del total de palabras de cada reseña. Se excluyeron de este proceso las palabras vacías, también conocidas como *stop words* en inglés, que son aquellas que no aportan un significado importante, ya que su presencia no contribuye de manera significativa a la comprensión del contenido.

Aplicando esta técnica tres veces, cada vez con un número distinto de palabras a sustituir: veinticinco por ciento del total, cuarenta y cinco por ciento del total y finalmente sesenta y cinco por ciento del total, se generaron tres nuevos conjuntos de datos, cada uno con el mismo número de reseñas que el conjunto original. Seguidamente, se procedió a hacer un análisis de los mismos para ver su similitud léxica y semántica respecto a los textos originales. Se pudo observar que los textos generados mantenían el significado semántico, pero a penas variaban el léxico. Esto último se

achacó a la limitación de la librería `Wordnet` en español. Aún así, al igual que en la retrotraducción, se generó el conjunto de datos definitivo, obteniendo los textos de mayor similitud semántica y menor similitud léxica de los tres conjuntos.

3.3.2.3. Inserción aleatoria

La inserción aleatoria consiste en insertar en posiciones aleatorias en el texto n sinónimos de palabras seleccionadas al azar, intentando evitar las palabras vacías.

Texto original:

texto = La estación está sucia porque no la limpian.

Se convierte en:

texto = La estación está sucia porque mugrienta no la limpian.

Figura 3.6: Inserción aleatoria

Para poder realizar la inserción aleatoria se hizo uso de la librería de `Wordnet` ya utilizada en la técnica de reemplazo por sinónimos. A pesar de las limitaciones de `WordNet` para trabajar con sinónimos en español, se mantuvo como la opción más adecuada. Una vez resuelto el acceso a los sinónimos, el siguiente paso fue determinar la cantidad de inserciones a realizar por cada reseña. Se optó por insertar un diez por ciento del total de palabras de cada texto. Es importante destacar que el porcentaje se calcula sobre el total de palabras de la reseña, aunque las palabras vacías no se consideran candidatas para la inserción.

Siguiendo el proceso de las dos técnicas anteriores, se generaron tres nuevas colecciones de datos. En cada una de ellas, las palabras insertadas y sus posiciones eran distintas, lo que permitió generar datos más variados. Al igual que en los anteriores procesos, para cada reseña del conjunto original se seleccionó el mejor texto (mayor similitud semántica y menor similitud léxica) de los tres disponibles. Aunque los resultados obtenidos fueron mejores que los del reemplazo por sinónimos, se observó poca alteración en el léxico.

3.3.2.4. Intercambio aleatorio

El intercambio aleatorio se fundamenta en permutar n veces la posición de dos palabras dentro de una misma frase.

Al no requerir de trabajar con sinónimos, este método no genera tantos problemas. Se optó por intercambiar un diez por ciento del total de las palabras de cada reseña. Además, como en los métodos previos, solo se consideraron para el intercambio aquellas palabras que no fueran vacías. Esta decisión se tomó para minimizar la pérdida de sentido en las reseñas al realizar los cambios.

Al igual que las técnicas pasadas, se generaron tres conjuntos de datos con el propósito de obtener un número considerables de reseñas distintas. En este caso, al

Texto original:

texto = La estación está sucia porque no la limpian.

Se convierte en:

texto = La estación está limpian porque no la sucia.

Figura 3.7: Intercambio aleatorio

realizar un análisis de los datos, se observaron ligeras variaciones en la semántica. Sin embargo, la similitud léxica siempre era uno ya que las palabras del texto original y del generado eran las mismas. Esto se debe a que la aplicación de esta estrategia simplemente reubica las palabras de un texto.

Finalmente, al igual que en las técnicas anteriores, se creó el conjunto de datos definitivo que recopilaba las reseñas más significativas generadas por este procedimiento.

3.3.2.5. Eliminación aleatoria

El método de eliminación aleatoria se basa en suprimir n palabras de una frase, seleccionando de forma aleatoria aquellas que no sean palabras vacías.

Texto original:

texto = La estación está sucia porque no la limpian.

Se convierte en:

texto = La estación está porque no la limpian.

Figura 3.8: Eliminación aleatoria

Este método es el más sencillo de los presentados, ya que únicamente implica la eliminación de ciertas palabras al azar en cada reseña. Como se realizó en los métodos anteriores el número de eliminaciones realizadas en cada reseña vuelve a ser un diez por ciento del total (sin tener en cuenta las palabras vacías).

Siguiendo la metodología previamente establecida, se generaron tres nuevos grupos de datos con el fin de obtener un conjunto amplio y diverso de reseñas distintas. El análisis de estos nuevos datos reveló la mayor variación de las tres anteriores técnicas. Esto se debe a que la eliminación de palabras afecta tanto al significado del texto como al conjunto de palabras que se utiliza. Finalmente, se escogió una única reseña sintética por cada reseña original.

3.3.2.6. Combinación de técnicas

Se observó que tras aplicar los anteriores cuatro procedimientos (reemplazo por sinónimos, inserción aleatoria, intercambio aleatorio y eliminación aleatoria), los cambios en los textos generados eran mínimos. Por esta razón, se decidió aplicar las cuatro técnicas de forma combinada, con el objetivo de lograr modificaciones más significativas.

Siguiendo la metodología habitual, se generaron tres nuevos conjuntos de datos, esta vez combinando todas las técnicas en lugar de utilizarlas de forma aislada. Seguidamente, se realizó un análisis detallado de los resultados. En esta ocasión, se observó que los cambios eran mucho más prometedores, ya que se lograba una mayor variabilidad léxica mientras se mantenía una alta similitud semántica, cumpliendo así los objetivos principales del aumento de datos.

Finalmente, como en las técnicas anteriores, se recopilaron las reseñas con mayor similitud semántica y menor similitud léxica entre los conjuntos generados.

3.3.2.7. Albumentación

La técnica de *albumentación* consiste en reorganizar todas las frases de un texto, eliminando aquellas que se repitan.

Texto original:
texto = <frase1><frase2><frase3><frase3><frase4>
Se convierte en:
texto = <frase2><frase4><frase1><frase3>

Figura 3.9: Albumentación

Para implementar este método, cada reseña fue dividida en frases individuales (si contenían más de una). Posteriormente, estas frases se mezclaron para generar nuevas versiones de la misma reseña con idéntico contenido, pero en distinto orden. Este proceso se repitió tres veces, como se hizo en las técnicas previas, generando así tres nuevas colecciones de datos.

El análisis de estos datos reveló que la similitud semántica experimentaba ligeras variaciones, mientras que la similitud léxica permanecía inalterada. Esto se debe a que no se introducen nuevas palabras ni se eliminan palabras existentes, como ocurre en el método de intercambio aleatorio. Sin embargo, los resultados obtenidos en cuanto a similitud semántica fueron más consistentes con esta técnica, debido a que el significado general de las reseñas se conservaba mejor.

Nuevamente, como en las técnicas anteriores, se creó el conjunto de datos definitivo recopilando las reseñas con mayor similitud semántica y menor similitud léxica entre los conjuntos generados.

3.3.2.8. Uso de modelos preentrenados

La siguiente técnica aplicada para el aumento de datos consistió en utilizar modelos preentrenados capaces de generar frases semánticamente equivalentes pero con variaciones léxicas. Estos modelos, conocidos como modelos de parafraseo, incluyen ejemplos como T5, PEGASUS y BART.

Para generar modelos con estas características adaptados al entorno del transporte público, se aplicó un método experimental riguroso basado en el método científico, para lo cual se siguieron los siguientes pasos: primero, se investigaron y comprendieron las características de cada conjunto de datos del que se disponía. A continuación, se definió que conjunto de datos o combinación de los mismos podría ser más adecuada. Se procedió entrenando los modelos con distintas configuraciones, tanto de parámetros propios del entrenamiento de modelo de procesamiento de lenguaje natural (número de capas congeladas, número de épocas ...) como del uso de distintos datos de entrenamiento. Finalmente, se evaluaron los modelos y se escogieron los que mejor resultado generasen.

GPT-2

OpenAI es una empresa estadounidense de investigación y despliegue de inteligencia artificial fundada en 2015. Uno de los hitos más conocidos de esta empresa es el desarrollo de la serie de modelos GPT (Generative Pre-trained Transformer). Estos modelos están diseñados para generar texto de manera coherente. La primera versión, GPT-1, fue lanzada en 2018 y se basa en la arquitectura Transformer. Tiene un tamaño de ciento diez y siete millones de parámetros, lo cual es bastante pequeño comparado con sus sucesores. Esto se debe a que GPT-1 fue un modelo de prueba para validar la idea de preentrenar un modelo de lenguaje y después afinarlo para tareas específicas sin necesidad de entrenar desde cero el modelo.

GPT-2 es el segundo modelo de la serie GPT. Lanzado en dos mil diez y nueve representó un avance significativo respecto al GPT-1. GPT-2, que mantiene una arquitectura de Transformer, tiene mil quinientos millones de parámetros y fue entrenado con grandes volúmenes de texto para predecir la siguiente palabra de una secuencia. Fue entrenado en una vasta cantidad de datos de internet, lo cual le permitió aprender sobre una amplia variedad de temas.

Dado que GPT-2 fue el último modelo completo lanzado como *open-source* por OpenAI, fue el modelo que se escogió para diseñar el sistema de parafraseo en cuestión. Debido a la limitación de hardware, se utilizó una versión ligera en castellano (`mrm8488/spanish-gpt2`), obtenida de Hugging Face. Con el propósito de adaptar el modelo al conjunto de datos sobre el que se trabajó, se realizó la técnica conocida como *fine-tuning* (ajuste fino). Este proceso consiste en adaptar un modelo previamente entrenado a tareas o casos de uso específicos. Concretamente, siguiendo este procedimiento, se generaron doce versiones del GPT-2 ligero. La mitad de estos modelos se ajustaron con el conjunto de datos original, sin tener en cuenta las etiquetas. Sin embargo, la segunda mitad de los modelos, siguiendo lo indicado en el artículo Data Augmentation Using Pre-trained Transformer Models, se ajustaron antepo-

niendo la etiqueta al texto correspondiente. Cada uno de los modelos generados fue ajustado en seis o tres *epochs* y ajustando el cien, cincuenta o veinte por ciento de las capas. Desafortunadamente, ninguno de los modelos fue capaz de parafrasear los datos de entrada, únicamente completaban las frases proporcionadas.

El fracaso de estos modelos podía deberse a varias razones: el modelo base era demasiado ligero, el *fine-tuning* se realizó utilizando pocos datos o la organización de los datos no era la correcta. El primero de los problemas no tenía solución ya que el "hardware" del que se disponía no era capaz de procesar modelos con más parámetros. El segundo problema tampoco tenía solución dado que no se disponía de más datos y no se podían obtener más. Pero, la tercera posibilidad si tuvo solución: para que el modelo aprendiese a parafrasear, los datos debían ser organizados por pares (frase original y frase parafraseada).

Dado que se habían aplicado varias técnicas de aumento de datos, se disponían de un gran número de parafraseos para cada texto del conjunto de datos original. De todos los conjuntos disponibles, aquellos que generaban textos más variados manteniendo la similitud semántica eran aquellos generados por la técnica de retrotraducción. Teniendo esto en cuenta, se decidió usar las tres colecciones de datos generadas por esta técnica. Consecuentemente, llegados a este momento, se disponían de tres pares de frases para cada texto del conjunto original (un par por cada variación de la retrotraducción).

Para entrenar la versión ligera de GPT-2 en castellano con los pares de textos disponibles, fue necesario añadir una columna más a la estructura de los datos. Es decir, el conjunto de datos estaba organizado de la siguiente forma: cada fila tenía dos columnas, una con la frase original y la otra con el parafraseo (texto generado por el método de la retrotraducción). Al igual que en el primer enfoque, se generaron dos colecciones de datos distintas para el entrenamiento: la primera sin añadir las etiquetas y la segunda anteponiendo las etiquetas a los textos. Originalmente, las etiquetas eran "v" (para las reseñas válidas) y "n" (para las reseñas inválidas). Sin embargo, para proporcionar más contexto semántico al modelo, "v" se extendió a "válida" y "n" a "noValida".

Tal y como se hizo en la primera versión del ajuste fino de GPT-2, se dividieron ambas colecciones de datos en conjuntos de entrenamiento (ochenta por ciento del total), validación (diez por ciento) y test (diez por ciento).

Para realizar el ajuste fino del modelo, el primer paso fue añadir al *tokenizador* los *tokens* de *padding* (<|pad|>), inicio del *prompt* (<start>), separación de la frase de entrada y parafraseo (<sep>) y por último el *token* de final de generación (<end>). Tal y como se realizó en el primer enfoque, se generaron doce versiones, la mitad de las cuales se ajustaron con el conjunto de datos original, sin tener en cuenta las etiquetas, y la segunda mitad con las etiquetas al inicio del texto. Cada uno de los modelos generados fue ajustado en seis o tres *epochs* y congelando el cero, cincuenta u ochenta por ciento de las capas.

Al igual que ocurrió con los primeros modelos, ninguna de las versiones generadas con los datos organizados en pares de frases era capaz de realizar parafraseos. Las doce versiones se limitaban a completar el texto de entrada. Además, la mayor

parte de los textos generados no tenían mucho sentido y muchas veces el significado se veía alterado. Por esta razón, se optó por aplicar la técnica de ajuste fino sobre otro modelo.

T5

T5 (Text-To-Text Transfer Transformer) es un modelo de lenguaje basado en la arquitectura Trasformer (la misma que se usa en otros modelos como `BERT` o `GPT`) desarrollado por Google Research en dos mil diez y nueve. T5 ha sido diseñado bajo el concepto de que todas las tareas de NLP pueden formularse como un problema "texto a texto". Esto significa que, en lugar de tener modelos separados para cada tarea (por ejemplo, clasificación, generación de texto, traducción, etc), T5 toma todas estas tareas y las transforma en una tarea común de "entrada de texto" a "salida de texto".

T5 fue preentrenado con un gran corpus de texto (*Colossal Clean Crawled Corpus, C4*) para aprender representaciones generales del lenguaje (sintaxis, semántica y relaciones contextuales). Sin embargo, las versiones de T5 estándares proporcionadas por Google no están pensadas para usarse en castellano. Consecuentemente, se decidió usar la versión mejorada de este modelo conocida como T5 que tiene el mismo número de parámetros que el T5 original pero se le realizó *fine-tuning* en más de mil tareas adicionales abarcando más idiomas (entre ellos el castellano). Debido a limitaciones de hardware el equipo se vio obligado a usar la versión "small" con muchos menos parámetros que el modelo base.

Se observó que para entrenar un modelo que tuviese una precisión considerable, los datos de los que se disponía eran insuficientes. Por esta razón, se optó hacer uso del conjunto de datos PAWS-X (Paraphrase Adversaries from Word Scrambling - Cross-lingual) de Google, una extensión del conjunto de datos PAWS pero adaptado a seis idiomas adicionales.

PAWS-X está diseñado para evaluar la capacidad de modelos de lenguaje en la detección de paráfrasis. Está compuesto por pares de oraciones que pueden ser paráfrasis o no.

Para evaluar la calidad de distintos modelos, se generaron tres conjuntos de datos. El primero incluyó únicamente los pares de frases de PAWS-X en castellano que fueran paráfrasis, eliminando aquellos que no lo eran. El segundo conjunto se construyó a partir de pares de frases obtenidos mediante retrotraducción, generando tres variaciones por cada frase original. Por último, se creó un tercer conjunto de datos que resultó de la combinación de los dos anteriores.

Se procedió entrenando seis modelos distintos: se entrenaron modelos usando cada uno de los tres conjuntos de datos disponibles y variando el porcentaje de neuronas congeladas. Desgraciadamente, ninguno de los modelos fue capaz de generar paráfrasis de calidad. Para cada frase que se le proporcionaba generaba frases idénticas o sin ningún sentido.

Como potencial mejora, se decidió introducir los pares de frases del conjunto de datos PAWS-X que no fuesen paráfrasis. Para que el modelo pudiese inferir que estas

frases no eran las que el modelo debería generar se penalizaron las salidas generadas durante el entrenamiento del modelo. Es decir, los siguientes modelos se entrenaron con ejemplos incorrectos junto con los correctos, ayudándolo a distinguir entre una paráfrasis válida y una no válida.

Sin embargo, el resultado obtenido no fue el esperado, debido a que los modelos seguían generando salidas muy parecidas a las que generaban sin haber añadido los ejemplos negativos. Por esta razón, se volvió a optar por usar otro modelo para volver a aplicar la técnica de *fine-tuning* e intentar obtener unos resultados mejores que con los dos previos modelos.

DeepSeek-R1

DeepSeek es una *startup* china de inteligencia artificial que ha ganado notoriedad por desarrollar modelos de lenguaje avanzados, entre ellos DeepSeek-R1. La introducción de este modelo tuvo un impacto considerable en el mercado de la inteligencia artificial, provocando una disminución notable en el valor de mercado de competidores estadounidenses.

DeepSeek-R1 es un modelo de razonamiento, que se diferencia de modelos de lenguaje tradicionales en que antes de generar una respuesta busca estructurar pasos intermedios de razonamiento. Este modelo fue entrenado utilizando técnicas de aprendizaje por refuerzo a gran escala, lo cual permitió que el modelo desarrollara comportamientos de razonamiento complejos de manera emergente.

DeepSeek hizo públicos los modelos, por lo que el equipo fue capaz de utilizar modelos de última generación para generar datos sintéticos. A diferencia de los otros modelos probados, DeepSeek-R1 fue compatible con la librería `Unsloth`, que permite realizar *fine-tuning* a modelos aproximadamente el doble de rápido y reduciendo hasta en un setenta por ciento el consumo de memoria sin degradación en la precisión del modelo. Debido a esta librería el tamaño del modelo no fue una limitación tan grande como en los anteriores dos casos. Consecuentemente, el modelo que se escogió fue: `DeepSeek-R1-Distill-Llama-8B-unsloth-bnb-4bit`, el cual es un modelo destilado (proceso por el cual los patrones de razonamiento de el modelo más grande original pueden "comprimirse" en modelos más reducidos con bastante precisión) y cuantizado a cuatro bits para reducir el tamaño del modelo y el uso de memoria.

Como se realizó con los anteriores modelos, se generó un conjunto de datos formado por PAWS-X en castellano solo con los ejemplos positivos (pares de frases que si son paráfrasis la una de la otra) y el conjunto de datos formado por los pares de frases de la retrotraducción.

Para orientar eficazmente a un modelo de IA y conseguir que obtenga respuestas precisas y relevantes es fundamental que el *prompt* tenga las siguientes características: debe ser un *prompt* claro y detallado, evitando ambigüedades, debe proporcionar un contexto adecuado y se debe definir la audiencia. Consecuentemente, el *prompt* diseñado fue: *Eres un experto en generación de datos sintéticos. Dado un texto de entrada, genera una paráfrasis que conserve el significado original pero utilice un vocabulario diferente y una estructura gramatical variada. El resultado debe ser útil*

para entrenar redes neuronales con datos sintéticos.

Finalmente, se entrenó el modelo con un cinco pasos de preentrenamiento para la tasa de aprendizaje, un número de pasos de ciento veinte y una tasa de aprendizaje de 2×10^{-4} . En cada paso se mostraron las siguientes perdidas en el entrenamiento (solo se muestran los pasos representativos):

Step	Training Loss
1	2.615600
2	2.569400
3	2.580000
4	2.361600
5	2.145300
6	1.745900
7	1.087500
113	0.569700
120	1.022800

Figura 3.10: DeepSeek-R1 Datos Entrenamiento 1

Como se puede ver, el descenso inicial es muy acelerado, lo cual es normal al comenzar a ajustar los pesos del modelo. Sin embargo, a lo lardo del entrenamiento la pérdida oscila entre 0.6 y 1.3, alcanzando un mínimo en el paso 113 y volviendo a subir hasta el último paso. Estas oscilaciones son indicadores de una tasa de aprendizaje alta que genera que el optimizador "salte" sobre mínimos locales. Consecuentemente, se entrenó otro modelo con una tasa de aprendizaje más reducida y un mayor número de pasos.

Con una configuración de la tasa de aprendizaje de 5×10^{-5} y 150 pasos, se obtuvieron los siguientes datos:

Step	Training Loss
1	0.745500
2	0.732100
3	0.859000
4	0.688800
5	0.963900
6	0.932600
7	0.573400
125	0.661400
143	0.683000

Figura 3.11: DeepSeek-R1 Datos Entrenamiento 2

Estos datos siguen indicando fluctuaciones, pero al final se estabilizan sin mejoras significativas, lo cual puede ser un indicador de *overfitting*. Consecuentemente, para poder realizar un estudió más completo se decidió mostrar también la pérdida en el conjunto de validación.

Tras muchas configuraciones de entrenamiento, los mejores resultados obtenidos son los presentados en la figura 3.12.

Step	Training Loss	Validation Loss
10	0.469100	11.189466
20	0.411100	11.150325
30	0.531800	11.136603
40	0.521300	11.152895
50	0.444200	11.170336
60	0.555600	11.186615
70	0.534200	11.188937
80	0.962000	11.204674
90	0.561300	11.212711
100	0.537900	11.229830
110	0.689000	11.229006
120	0.980000	11.229245
130	0.711900	11.230718
140	0.657800	11.229288

Figura 3.12: DeepSeek-R1 Datos Entrenamiento 3

Los resultados mostrados demuestran que el modelo estaba sobre-ajustando y generalizando. Se llegó a la conclusión de que la principal causa de estos resultados se debía a la combinación del uso de un modelo ligero y una tarea muy compleja. Aunque el modelo en cuestión tuviese ocho mil millones de parámetros, la versión destilada y cuantizada a 4 bits tiene una capacidad efectiva menor. La compresión numérica reduce la precisión en la representación de los pesos, lo que puede limitar la habilidad del modelo para captar las sutilezas y complejidades del lenguaje necesarias para generar parafraseos de alta calidad.

Aunque DeepSeek-R1 generó los mejores resultado de modelos ligeros a los que se les realizaron *fine-tuning*, esta tarea requería entender matices semánticos y estructurales del lenguaje muy complejos para un modelo de estas características. Se llegó a la conclusión de que debido a las limitaciones del modelo, éste solo estaba aprendiendo bien los patrones del conjunto de entrenamiento, lo cual genera un *Training Loss* bajo, pero no generalizaba correctamente nuevos ejemplos, resultando en un *Validation Loss* persistentemente alto.

3.3.2.9. Uso de modelos de última generación vía API

Debido a las limitaciones hardware del equipo, se pudo comprobar que el *fine-tuning* de redes neuronales para generación de paráfrasis no era viable.

Como último método para la generación de datos sintéticos, se optó por utilizar modelos de última generación a través de sus respectivas API. De esta manera, no fue necesario realizar un proceso de *fine-tuning*, ya que las redes neuronales ofrecidas por empresas como OpenAI, Google y Anthropic son lo suficientemente sofisticadas para llevar a cabo la tarea sin necesidad de un refinamiento previo.

En particular, se decidió utilizar los modelos proporcionados por OpenAI. La primera decisión a tomar fue la elección del modelo adecuado. Los modelos de lenguaje disponibles se pueden dividir en dos categorías: los modelos de razonamiento, que han sido entrenados mediante aprendizaje por refuerzo para abordar tareas complejas y, antes de generar una respuesta, estructuran una serie de pasos de razonamiento; y los modelos clásicos de lenguaje, que carecen de esta capacidad de razonamiento y responden de manera directa sin generar una cadena de pensamientos previos.

Debido a que la tarea de generación de paráfrasis no requiere descomponer un problema en pasos lógicos complejos, sino que se centra en reestructurar y reformular oraciones manteniendo un mismo significado, se tomó la decisión de limitar el modelo a la categoría de los no razonadores. De todos los modelos ofrecidos, se escogió GPT-4o-mini ya que ofrece un gran nivel de inteligencia, genera resultados a una alta velocidad y es uno de los modelos más asequibles.

Aunque un modelo de estas características no necesite ser ajustado para realizar una tarea de este calibre, el mensaje del sistema y el *prompt* tienen que ser cuidadosamente diseñados para que GPT-4o-mini sea capaz de generar la salida correcta y en el formato deseado.

Los modelos de chat-GPT (y muchos otros) funcionan mediante el intercambio de mensajes. El usuario manda un mensaje al modelo, el modelo lo recibe, procesa la información y responde mediante otro mensaje. En concreto, los mensajes de OpenAI son un diccionario con al menos dos claves: el rol (*system, user assistant*) y el contenido del mensaje.

El mensaje del sistema (*system message*), es el primer mensaje que recibe el modelo y sirve para darle instrucciones o darle contexto al modelo. Este mensaje, al que se le da un formato distinto al resto, guía el comportamiento del modelo, determina el tono, especifica el formato deseado de salida del modelo ... Utilizando el mensaje del sistema de manera efectiva se puede conseguir que el modelo genere respuestas más concretas y relevantes. Teniendo esto en cuenta, se generó el siguiente mensaje:

Eres un experto en procesamiento del lenguaje natural con habilidades avanzadas en reformulación de textos. Tu tarea es parafrasear un texto de manera que el significado se mantenga intacto, pero con cambios sustanciales en la estructura sintáctica y el vocabulario. Usa sinónimos, reformulaciones y construcciones alternativas para evitar repeticiones. Evita cambiar el tono y la intención del mensaje original. Si el texto contiene términos técnicos o nombres propios, consérvalos sin cambios. No agregues ni elimines información que altere el significado original. La respuesta debes proporcionarla en una linea

Como se pude ver, se le indicó claramente al modelo cual era su rol, que tarea iba realizar, cuanto podía manipular el texto de entrada (usar sinónimos, conservar los nombres propios o términos técnicos ...) y como debía ser el texto de salida.

Únicamente un diseño riguroso del mensaje del sistema no es suficiente. También se debe diseñar el prompt cuidadosamente para que el modelo realice la tarea deseada. Para ello, el prompt disponía de las siguientes características: instrucción clara, limitaciones, variaciones recomendadas y formato de salida.

Parafrasea el siguiente texto asegurando una alta variabilidad léxica y sintáctica, pero conservando el significado original: ***Texto a parafrasear*** *Usa sinónimos y expresiones equivalentes. Cambia la estructura de las oraciones sin perder claridad. Mantén el tono y la intención del mensaje. No agregues ni elimines información clave. Asegúrate de que la reformulación sea natural y fluida. Asegúrate de que la respuesta está toda en una linea. Respuesta:*

Para cada frase del conjunto de datos original se generó una paráfrasis. El resultado fue muy bueno comparado con las otras técnicas de aumento de datos y el tiempo requerido fue reducido.

3.3.3. Conclusiones del aumento de datos

Aunque la creación y utilización de datos sintéticos suponga un reto y un riesgo, en este caso se determinó que como necesario debido a la reducida cantidad de datos disponibles.

Sin duda alguna, los métodos que mejores resultado generaron fueron la retrotraducción y la utilización de modelos de lenguaje avanzados vía API (GPT-4o-mini). Estos métodos generaban paráfrasis de mayor calidad. Es decir, generaban frases semánticamente similares a la original pero con variabilidad léxica y gramatical. Aunque las técnicas de reemplazo por sinónimos, inserción aleatoria, intercambio aleatorio y eliminación aleatoria, por si solas fuesen insuficientes, la combinación de todas ellas también generaron resultados aceptables.

Tras aplicar todo este proceso, el conjunto de datos se multiplicó por cinco. Lo cual supuso un factor diferencial para entrenar un clasificador con mejor precisión.

3.3.4. Problemas con el aumento de datos

Cuando se inició el proceso de retrotraducción, el equipo se dio cuenta que las reseñas que contenían algún tipo de "emoji" no podían traducirse de forma correcta. Se propusieron varias soluciones: eliminar aquellas reseñas que contuviesen algún "emoji" (este enfoqué se descartó debido a que se perdería información importante), cambiar los "emoji" por texto que los describieran, lo cual también fue descartado debido a que las reseñas que se generarían podrían carecer de sentido (Ej: "El personal de la estación es muy desagradable dedo gordo hacia abajo"), y por último, la solución que finalmente se aplicó, eliminar los "emojis" de las reseñas y eliminar aquellas que se quedarían vacías.

Originalmente, aquellas reseñas que solo contuviesen "emojis" se podrían haber considerado válidas, pero tras aplicar este método, se considerarían inválidas o se eliminarían al quedarse vacías. Como la mayor parte de las reseñas no contenían este tipo de caracteres, los datos a penas se vieron afectados (solo ciento treinta y seis reseñas cambiaron y dos fueron eliminadas).

La principal limitación durante este proceso fue el hardware. Los modelos necesarios para llevar a cabo una tarea con esta complejidad, requerían de gráficas

superiores a las RTX 3060 y RTX 3080, las únicas disponibles para el equipo.

3.3.5. Diseño del clasificador de reseñas

El objetivo del clasificador era clasificar las reseñas disponibles en dos clases: útiles (aquellas reseñas que luego se utilizarían para extraer las características de cada estación de metro) e inútiles (las que no dispusiesen de información relevante para la extracción de información de las estaciones). Es decir, la tarea en cuestión era el desarrollo de un clasificador binario.

El primer paso consistió en escoger el modelo que llevaría a cabo esta clasificación. El algoritmo de Naive-Bayes supone que las palabras en un texto son independientes entre sí, y que la posibilidad de que un texto pertenezca a una clase es proporcional al producto de las probabilidades de cada palabra en esa clase. Para usar este modelo, primero se deben convertir los textos en vectores de recuento de palabras o frecuencias para luego aplicar el Teorema de Bayes y calcular las probabilidades de clase. Este modelo se descartó debido a la complejidad de las reseñas. Por esta misma razón no se aplicó una regresión logística.

Considerando el número limitado de reseñas para el entrenamiento y su longitud y complejidad, se decidió usar modelos preentrenados de lenguaje natural. En concreto, se utilizó XLM-RoBERTa, una versión mejorada de BERT desarrollada por Facebook AI (Meta AI) en dos mil diez y nueve. Este modelo fue preentrenado en dos terabytes y medio de datos filtrados de CommonCrawl conteniendo más de cien idiomas. Es un modelo de Transformers entrenado de forma auto supervisada. Es decir, fue preentrenado en textos sin etiquetas asignadas. Fue entrenado con el objetivo de *Mask Language Modeling* (MLM): dada una frase, el modelo de forma aleatoria enmascaraba el quince por ciento de las palabras de entrada y tenía como objetivo predecirlas.

El primer paso fue definir qué conjunto de información utilizar. En la etapa anterior del proyecto se llevó a cabo una exhaustiva investigación sobre diversas técnicas de aumento de datos textuales. Como resultado, para este punto ya se contaba con una cantidad significativa de textos destinados al entrenamiento. Con el objetivo de evaluar el impacto específico de los datos aumentados, se optó por agruparlos según la estrategia empleada: datos originales, retrotraducción, generación mediante grandes modelos de lenguaje, combinación de técnicas simples (reemplazo por sinónimos, inserción aleatoria, intercambio aleatorio y eliminación aleatoria), y albumentación.

Se implementaron tres estrategias de entrenamiento: la primera consistió en una fase con datos sintéticos y originales juntos; la segunda, en entrenamiento dividido en dos fases: donde la primera se realizó con datos sintéticos y la segunda con los datos originales y la tercera, un entrenamiento exclusivo con datos originales.

3.3.5.1. Entrenamiento en una fase con datos originales y sintéticos

En este caso, se generó un nuevo conjunto de datos compuesto por todos los textos disponibles, tanto los originales como los generados mediante el aumento de

datos. Como se puede ver en la figura 3.13, el entrenamiento se llevó a cabo utilizando una tasa de aprendizaje de 5×10^{-6} y un tamaño de lote de ocho muestras. La evaluación se realizaba al final de cada época, tomando la exactitud (*accuracy*) como métrica principal para seleccionar el mejor modelo. Asimismo, se implementó una estrategia de parada temprana (*early stopping*), que detendría el entrenamiento automáticamente si no se observaban mejoras significativas en el rendimiento. Específicamente, si durante tres épocas consecutivas no se registraba una mejora, el proceso de entrenamiento se interrumpía.

```
modelName = 'FacebookAI/xlm-roberta-base'

earlyStoppingCallback = EarlyStoppingCallback(
    early_stopping_patience=3,
    early_stopping_threshold=0.01,
)

trainingSavingPath = "../../1. Models/Clasificador/Training1"
finalSavingPath = "../../1. Models/Clasificador/Results1"

trainingArgs = TrainingArguments(
    output_dir = trainingSavingPath,
    num_train_epochs = 20,
    learning_rate = 5e-6,
    per_device_train_batch_size = 8,
    per_device_eval_batch_size = 8,
    eval_strategy = "epoch",
    save_strategy = "epoch",
    load_best_model_at_end = True,
    metric_for_best_model = "accuracy",
)
```

Figura 3.13: Parámetros de entrenamiento con datos originales y sintéticos en una fase

Considerando la poca cantidad de datos disponibles, como se observa en la figura 3.14, los resultados obtenidos fueron bastante positivos. Sin embargo, el número de muestras correspondientes a la clase inválida era considerablemente menor que el de la clase válida, lo cual afectaba la evaluación del modelo. En cuanto a las métricas, la precisión, que indica la capacidad del modelo para evitar falsos positivos, fue mayor para la clase válida, lo que evidencia que el modelo realiza predicciones más acertadas para esta categoría. Del mismo modo, el *recall*, que mide la capacidad para identificar verdaderos positivos, también fue más alto en la clase válida. Como consecuencia, el *F1-score*, que representa el promedio armónico entre precisión y *recall*, era superior para esta clase, como era esperable dadas las métricas individuales.

La matriz de confusión correspondiente a este modelo (3.15) reforzaba lo anterior: el modelo funcionaba mejor para la clase válida, mientras que la clase inválida presentaba más errores, especialmente falsos negativos.

En conclusión, aunque el modelo obtuvo resultados satisfactorios, mostraba indicios de sesgo hacia la clase mayoritaria.

Classification Report:

	precision	recall	f1-score	support
inválida	0.82	0.79	0.81	615
válida	0.88	0.90	0.89	1005
accuracy			0.86	1620
macro avg	0.85	0.84	0.85	1620
weighted avg	0.86	0.86	0.86	1620

Figura 3.14: Informe del modelo entrenado con datos originales y sintéticos en una fase

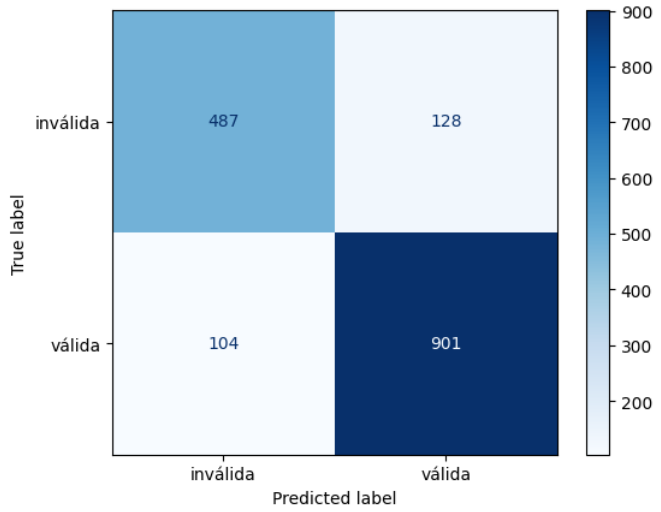

Figura 3.15: Matriz de confusión del modelo entrenado con datos originales y sintéticos en una fase

3.3.5.2. Entrenamiento en dos fases con datos originales y sintéticos

Aunque se emplearon los mismos datos que en el modelo anterior (tanto los originales como los generados mediante técnicas de aumento), la estrategia de entrenamiento fue distinta. En una primera etapa, orientada a la adaptación inicial del modelo al dominio y al estilo de las reseñas, se utilizaron únicamente los datos sintéticos. Posteriormente, en una segunda fase de refinamiento, se entrenó el modelo con los datos reales, con el fin de ajustarlo a la distribución y complejidad auténticas del conjunto.

En ambas fases se aplicaron las mismas métricas de evaluación que en el modelo previamente descrito, incorporando además la misma estrategia de parada temprana, que permitía detener el entrenamiento en ausencia de mejoras significativas.

Tal y como se observa en las figuras 3.16 y 3.17, los resultados obtenidos fueron muy similares a los del modelo anterior, aunque ligeramente inferiores, con una disminución aproximada del dos por ciento en las métricas generales. Este segundo

Classification Report:

	precision	recall	f1-score	support
inválida	0.79	0.81	0.80	615
válida	0.88	0.87	0.87	1005
accuracy			0.84	1620
macro avg	0.83	0.84	0.83	1620
weighted avg	0.84	0.84	0.84	1620

Figura 3.16: Informe del modelo entrenado con datos originales y sintéticos en dos fases

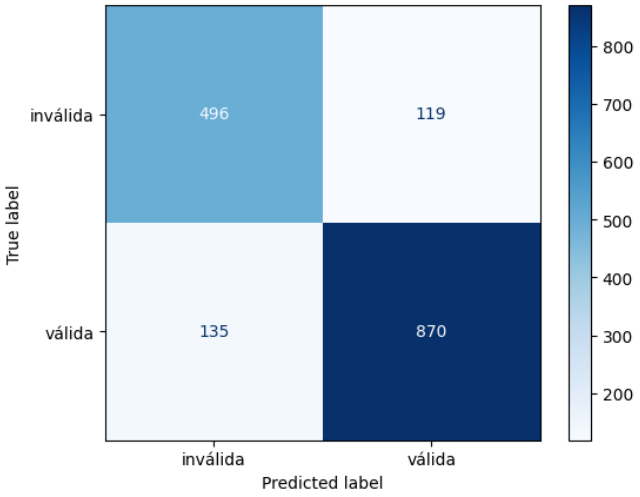

Figura 3.17: Matriz de confusión del modelo entrenado con datos originales y sintéticos en dos fases

enfoque no logró mejorar el rendimiento en ninguna de las clases evaluadas.

3.3.5.3. Entrenamiento en una fase con datos originales

Con el objetivo de evaluar la utilidad real de los datos sintéticos, se entrenó un modelo utilizando únicamente los datos originales. El procedimiento seguido fue equivalente al empleado en los clasificadores entrenados en una sola fase. Se mantuvieron constantes tanto los parámetros de entrenamiento como los criterios de evaluación, utilizando los mismos datos y métricas aplicados en los modelos anteriores.

La reducción significativa en la cantidad de datos de entrenamiento permitió acelerar notablemente el proceso. No obstante, como se observa en las figuras 3.18 y 3.19 , aunque las métricas obtenidas fueron similares a las de los modelos anteriores, presentaron un descenso constante, aproximadamente del cinco por ciento. A pesar de ello, considerando el volumen limitado de datos disponibles (en torno a

Classification Report:

	precision	recall	f1-score	support
inválida	0.76	0.75	0.75	615
válida	0.85	0.85	0.85	1005
accuracy			0.81	1620
macro avg	0.80	0.80	0.80	1620
weighted avg	0.81	0.81	0.81	1620

Figura 3.18: Informe del modelo entrenado con datos originales en una fase

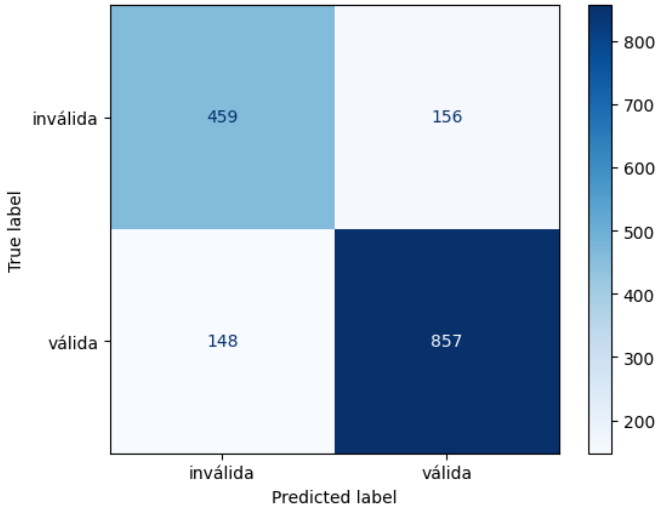

Figura 3.19: Matriz de confusión del modelo entrenado con datos originales en una fase

tres mil quinientos ejemplos), los resultados fueron sorprendentemente competitivos. Esto constituye una evidencia clara del potencial de los grandes modelos de lenguaje, cuya capacidad para captar estructuras complejas del lenguaje natural resulta fundamental incluso en contextos de datos reducidos.

3.3.5.4. Selección del modelo y procesamiento de los datos

Una vez entrenadas las tres variantes del clasificador, el siguiente paso consistía en seleccionar el modelo con mejor rendimiento. Si bien ninguno alcanzó el nivel de desempeño ideal, el modelo entrenado con datos sintéticos y originales en una sola etapa se posicionó como el claro ganador. Esta decisión se fundamentaba en que, aunque todos los modelos mostraban un mejor comportamiento al clasificar reseñas etiquetadas como válidas, el primero presentaba la menor diferencia en precisión entre ambas clases y, en términos generales, obtenía las mejores métricas.

Seleccionado el modelo, se procedió a clasificar la totalidad de las reseñas recopiladas en la fase inicial del proyecto, correspondientes a todas las estaciones de

la red de metro de Madrid. Como resultado, se generó un nuevo conjunto de datos compuesto exclusivamente por aquellas reseñas clasificadas como válidas, a partir de las cuales se extraerían posteriormente las características asociadas a cada estación.

3.4. Extracción de características

Una vez clasificadas todas las reseñas y aisladas aquellas consideradas válidas, el siguiente paso consistió en extraer las características relevantes de cada una. Esta tarea implicaba obtener la información más significativa contenida en las reseñas asociadas a cada estación, con el objetivo de construir una representación útil que permitiera generar recomendaciones personalizadas para los usuarios. Para ello, se llevó a cabo un estudio comparativo de distintas técnicas de extracción de características, utilizando un conjunto de reseñas de prueba. Tras implementar y analizar cada enfoque, se seleccionó la técnica que ofrecía los mejores resultados.

3.4.1. Estudio de distintas técnicas

La primera técnica implementada fue la de los *N-gramas*. Un *N-grama* se define como una secuencia de n palabras consecutivas dentro de un texto. Esta metodología facilita la extracción de información relevante a partir de las reseñas, ya que permiten capturar patrones locales del lenguaje y expresiones recurrentes, aplicando distintos tamaños de *N-gramas*.

Como es habitual en este tipo de enfoques, el primer paso consistió en eliminar las *stopwords*, es decir, aquellas palabras que carecen de contenido semántico relevante. A continuación, y con el objetivo de maximizar la fiabilidad del proceso, se optó por utilizar *N-gramas* de entre dos y cinco palabras, abarcando así desde bigramas hasta pentagramas. Limitarse únicamente a bigramas habría implicado una pérdida considerable de contexto y, por tanto, de información valiosa. En cambio, trabajar exclusivamente con pentagramas podría haber introducido ruido o distorsiones debido al exceso de información contextual. Por ello, y a pesar del mayor coste computacional asociado, se adoptó una estrategia combinada que incorporaba múltiples tamaños de n-gramas, lo cual permitió obtener una mayor precisión en la extracción de características.

Para llevar a cabo este proceso se utilizó la biblioteca de Python `Scikit-learn`, concretamente el módulo `CountVectorizer`, que permitió vectorizar las reseñas generando los diferentes *N-gramas*. Una vez realizada la segmentación de las reseñas en secuencias de n palabras, se procedió a extraer, de cada *N-gramas*, los sustantivos junto con su adjetivo adyacente más próximo. Esta elección se fundamenta en la estructura gramatical del español, donde los adjetivos calificativos suelen aparecer inmediatamente junto al sustantivo al que modifican o en su proximidad, lo que facilita la identificación de descripciones relevantes.

Este procedimiento se llevó a cabo utilizando la biblioteca `spaCy`, una herramienta de procesamiento de lenguaje natural desarrollada por Explosion AI. `spaCy`

proporciona modelos preentrenados para diversos idiomas, incluido el español, y permite realizar tareas como análisis morfo-sintáctico, lematización, etiquetado gramatical (*POS tagging*) y reconocimiento de entidades nombradas (NER), a través de un sistema modular estructurado en un *pipeline* compuesto por componentes encadenados (tokenizador, etiquetador, analizador sintáctico, etc.).

En este caso, `spaCy` se empleó para identificar la clase gramatical de cada palabra dentro de los *N-Gramas* y filtrar aquellos que contenían combinaciones de sustantivo y adjetivo. Los resultados se almacenaron en un diccionario, donde cada clave correspondía a un sustantivo y su valor asociado consistía en la lista de adjetivos con los que aparecía en contexto.

Una vez aplicada la técnica clásica de extracción de información basada en *N-Gramas*, se procedió a explorar enfoques más avanzados mediante el uso de bibliotecas basadas en redes neuronales. Si bien el preprocesamiento requerido por estas herramientas difería notablemente del utilizado en el enfoque anterior, la segunda etapa del proceso mantenía el mismo objetivo: identificar los distintos adjetivos asociados a los sustantivos correspondientes dentro de las reseñas, con el fin de extraer descripciones significativas.

La primera de las bibliotecas neuronales exploradas fue nuevamente `spaCy`, aunque en este caso se utilizó con un enfoque diferente. Concretamente, se aprovecharon sus capacidades de análisis de dependencias sintácticas para identificar las relaciones gramaticales entre las palabras dentro de cada reseña. `spaCy` permite determinar tanto la palabra de la que depende un término dado como las palabras que dependen de él, es decir, sus "hijos" sintácticos.

A partir de esta estructura, se construyó un nuevo diccionario donde cada clave correspondía a un sustantivo y el valor asociado era el conjunto de adjetivos relacionados gramaticalmente con dicho sustantivo. Además, `spaCy` permite identificar si un adjetivo es modificado por otras palabras, como adverbios o partículas de negación. Por ejemplo, en la frase "El metro va muy rápido", el adverbio "muy" modifica el adjetivo "rápido", intensificando su significado. Del mismo modo, en expresiones como "El tren no es nada lento", la secuencia "nada lento" transmite un significado opuesto al adjetivo aislado.

Para no perder este tipo de matices semánticos, se optó por incluir no solo los adjetivos, sino también las combinaciones de adjetivo con sus modificadores en el diccionario final. Esta decisión permitió conservar descripciones más precisas y representativas del contenido de las reseñas.

`Stanza`, la segunda librería basada en redes neuronales que se utilizó, fue desarrollada por un grupo de investigación de inteligencia artificial de la Universidad de Stanford. Destaca por su alto rendimiento en tareas de procesamiento de lenguaje natural, especialmente en el análisis sintáctico. Está disponible para una amplia gama de idiomas, entre ellos el español.

`Stanza` es una biblioteca basada en modelos de redes neuronales profundas, entrenados sobre grandes corpus lingüísticos, que ofrece un análisis detallado del texto. Entre sus funcionalidades se encuentran el etiquetado de partes del habla (*POS tagging*), el análisis de dependencias sintácticas y la lematización, lo que la convierte

en una herramienta avanzada para el análisis estructural de oraciones.

Dado que su funcionamiento es similar al de `spaCy`, se aplicó un procesamiento equivalente: a partir del análisis gramatical de las reseñas, se construyó un diccionario en el que cada clave correspondía a un sustantivo, y su valor asociado consistía en los adjetivos relacionados gramaticalmente con él. Esta estructura permitió mantener la coherencia en la extracción de características, utilizando distintas bibliotecas con enfoques neuronales complementarios.

3.4.2. Comparación de las distintas técnicas y selección de la más adecuada

Para realizar una comparación objetiva de las distintas técnicas, se aplicaron todas ellas sobre el mismo conjunto de reseñas (3.20). Como resultados, se obtuvieron tres diccionarios distintos, cada uno correspondiente a una de las metodologías empleadas, lo que permitió analizar los distintos sustantivos detectados y sus correspondientes adjetivos asociados.

```
reviews = [
    "La estación de metro es moderna y limpia, pero los trenes son lentos.",
    "A veces hay demasiada gente y el servicio es malo.",
    "El metro de Madrid es rápido y eficiente, aunque algunas estaciones están sucias.",
    "Buena conexión con otras líneas, pero los horarios nunca son confiables."
]
```

Figura 3.20: Reseñas de prueba para comparación de métodos de extracción de características

Como se aprecia en la figura 3.21, a técnica basada en *N-Gramas* presentaba varias limitaciones, entre ellas su incapacidad para capturar relaciones sintácticas a larga distancia dentro de una oración. Por ejemplo, un adjetivo situado al final de una frase puede estar calificando a un sustantivo ubicado al principio, una dependencia que los *N-Gramas* no pueden detectar de forma efectiva.

```
{'estación': {'sucio'},
 'conexión': {'confiable', 'horario', 'línea'},
 'gente': {'malo', 'servicio'},
 'servicio': {'malo'},
 'moderna': {'limpio'},
 'horario': {'confiable'},
 'tren': {'lento'},
 'línea': {'confiable', 'horario'},
 'metro': {'eficiente', 'moderno', 'rápido'}})
```

Figura 3.21: Diccionario obtenido con el método de los n-gramas para las reseñas de prueba

En contraste, las bibliotecas `spaCy` y `Stanza` (figuras 3.22 y 3.23) sí eran capaces de identificar este tipo de relaciones gracias a sus modelos de análisis de dependencias, lo que se tradujo en un rendimiento superior. No obstante, dado

```
{'estacion': {'limpio',
  'mal cuidado',
  'moderno',
  'mucho feo',
  'sucio'},
 'servicio': {'malo'},
 'metro': {'eficiente', 'rápido'},
 'conexion': {'buen'},
 'horario': {'mucho util', 'nunca confiable'}})
```

Figura 3.22: Diccionario obtenido con spaCy para las reseñas de prueba

```
{'estación': {'moderno', 'sucia'},
 'tren': {'lento'},
 'servicio': {'malo'},
 'metro': {'rápido'},
 'conexión': {'buena', 'confiable'},
 'horario': {'confiable', 'util'},
 'estacion': {'feo'}})
```

Figura 3.23: Diccionario obtenido con `Stanza` para las reseñas de prueba

que ambas herramientas ofrecían resultados muy similares, la diferencia decisiva radicó en las funcionalidades adicionales de `spaCy`. Como se mencionó anteriormente, `spaCy` permite acceder tanto a la palabra de la que depende un término como a las palabras que dependen de él (sus "hijos"), lo cual proporciona una representación más completa y detallada de la estructura gramatical de cada oración.

Por este motivo, y tras confirmar la equivalencia en el rendimiento general, se seleccionó `spaCy` como herramienta principal para el análisis lingüístico.

3.4.3. Obtención de todas las características

Una vez seleccionado el método de extracción, se amplió su funcionalidad para incluir un conteo de las apariciones de cada adjetivo relacionado a un sustantivo determinado. Esto se hizo para que posteriormente se pudiesen solucionar inconsistencias en las descripciones. Por ejemplo, es posible que una misma estación fuese descrita en las reseñas como "rápida" y "lenta", lo cual resulta contradictorio. El recuento permitiría eliminar el adjetivo menos frecuente en estos casos.

A continuación, utilizando las reseñas filtradas por el clasificador, se obtuvo un diccionario con sustantivos y sus correspondientes adjetivos con sus frecuencias para cada estación del Metro de Madrid. Por ejemplo, el resultado obtenido para la estación Alonso de Mendoza es el mostrado en la figura 3.24.

3.4.4. Combinación de sustantivos

Como se puede ver en la figura 3.24, el proceso de extracción genera un diccionario amplio y detallado, que recoge una gran variedad de sustantivos y adjetivos asociados. Sin embargo, en la variedad es donde se encuentra el desafío. Analizando

```
defaultdict(collections.Counter,
            {'parte': Counter({'negativo': 1}),
             'escalera': Counter({'mecánico': 1,
                      'ascensor': 1,
                      'vigilante': 1}),
             'interfono': Counter({'aposta': 1}),
             'estacion': Counter({'buen': 2,
                      'mucho concurrido': 1,
                      'lleno': 1,
                      'perfecto': 1,
                      'último': 1,
                      'inagurar': 1,
                      'mucho limpio': 1,
                      'bien': 1,
                      'buna': 1,
                      'genial': 1,
                      'buena': 1,
                      'moderno': 1}),
             'hora': Counter({'pico': 1}),
             'ayuda': Counter({'gran': 1}),
             'zona': Counter({'visto': 1, 'sur': 1}),
             'verde': Counter({'oscuro': 1}),
             'autovia': Counter({'junto': 1}),
             'aparcamiento': Counter({'amplio': 1}),
             'parada': Counter({'rápido': 1}),
...
             'cobertura': Counter({'móvil': 1}),
             'fria': Counter({'mucho impersonal': 1}),
             'acceso': Counter({'fácil': 1, 'buen': 1}),
             'horario': Counter({'indicado': 1}),
             'metro': Counter({'típico': 1})})
```

Figura 3.24: Resultado de aplicar el método de extracción de características sobre Alonso de Mendoza

varios resultados, se observó que varios sustantivos añadidos a los diccionarios tenían el mismo significado. En el caso específico de Alonso de Mendoza, se puede ver este problema en los sustantivos estación y parada, que claramente hacen referencia al mismo concepto.

Para hacer frente a este problema, se implementó un proceso de agrupación basado en la similitud entre palabras utilizando embeddings lingüísticos. Cuando la distancia entre dos sustantivos se encontraba por encima de un umbral determinado, se consideraban sinónimos y se fusionaban sus características.

Para llevar a cabo esta tarea, se emplearon dos modelos preentrenados disponibles en la plataforma *Hugging Face*, ambos especializados en la representación semántica de frases: `paraphrase-multilingual-MiniLM-L12-v2` y `LaBSE`, ambos pertenecientes a la familia de *Sentence-Transformers*. Dado que cada modelo arrojaba distancias diferentes para el mismo par de palabras, se realizaron múltiples pruebas con ambos para determinar un umbral óptimo de similitud. Si bien ninguno de los modelos ofrecía resultados completamente consistentes, ya que cada uno reconocía sinónimos distintos con mayor o menor precisión, se optó por calcular la media de las distancias obtenidas por ambos modelos, con el fin de suavizar las discrepancias y mejorar la estabilidad del agrupamiento.

Finalmente, tras diversas pruebas empíricas, se estableció un umbral de similitud de 0,65. Aunque este valor no es especialmente elevado, resultó ser suficientemente riguroso para identificar sinónimos de forma efectiva sin introducir fusiones incorrectas.

3.4.5. Combinación de adjetivos

Una vez colapsados los sustantivos identificados como sinónimos, se observó que el mismo problema surgía con los adjetivos. Es decir, para un mismo sustantivo podía darse el caso de obtener varios adjetivos distintos pero con mismo significado. Para solucionar este problema, se utilizó la misma función que en el caso anterior, solo que además se tuvieron que sumar las frecuencias de los sinónimos.

3.4.6. Eliminación de antónimos

El siguiente desafío que se abordó fue la eliminación de antónimos. El objetivo ere evitar que adjetivos con significados opuestos coexistieran en el conjunto de características de un mismo sustantivo, ya que esto podría introducir contradicciones en la caracterización de las estaciones. Sin embargo, tras una extensa búsqueda se llegó a la conclusión de que encontrar una solución eficiente para este problema resultaba especialmente complejo. La principal limitación era encontrar una herramienta útil para la detección de antónimos en castellano.

Ante esta limitación, se optó por utilizar la biblioteca `nltk`, ampliamente reconocida por su capacidad para manejar relaciones semánticas en inglés. Para aprovechar su funcionalidad, se tradujeron los adjetivos del español al inglés utilizando la librería `deep translator`. A partir de estas traducciones, se obtuvieron los antónimos mediante `nltk`, y posteriormente se tradujeron de vuelta al español. Este enfoque permitió identificar antónimos con un grado razonable de precisión. Sin embargo, debido a la gran cantidad de datos extraídos y los tiempos de espera de las APIs, el tiempo de procesamiento era demasiado alto y finalmente se optó por realizar la limpieza de los antónimos a mano. Para ello, se revisaron los adjetivos asociados a cada estación y, en los casos en que se detectaban pares de adjetivos con significados opuestos vinculados a un mismo sustantivo, se conservaba únicamente aquel con mayor frecuencia de aparición.

3.4.7. Filtrado de sustantivos

Al haber realizado la correspondiente limpieza de las características se pasó a analizarlas y seleccionar aquellas que fuesen útiles. Para poder llevar a cabo esto primero se optó por ver cuales eran los sustantivos que más veces aparecían en todas las reseñas y se realizó un conteo de los mismos. Una vez obtenido el conteo se apreció como había muchos sustantivos que no eran relevantes, es decir, no tenían una frecuencia muy alta (número de estaciones en las que aparece) y se decidió quedarse con aquellos que apareciesen en más de treinta estaciones.

3.4.8. Filtrado de adjetivos

Para simplificar las características, se eliminaron los intensificadores y negadores (como "mucho", "bien", "poco", "nunca", "no", entre otros). Para ello, se creó manualmente un diccionario en el que la clave correspondía a la característica con el modificador y el valor representaba su significado real sin dicho modificador. Esta estrategia se aplicó debido a que, en algunos casos, eliminar simplemente el modificador podía cambiar completamente el sentido de la característica. Por ejemplo, "no lento" se transforma semánticamente en "rápido", por lo que conservar solo "lento" induciría a error.

Una vez finalizado el diccionario, se procesó el conjunto original de características para aplicar los cambios correspondientes. Cuando varias adjetivos modificados se reducían a una misma característica, se sumaban sus frecuencias. Por ejemplo, si "no lento" se transformaba en "rápido", y si "rápido" ya existía en los datos, se acumulaba la frecuencia de ambas.

Para facilitar futuros procesamientos, se decidió que lo más adecuado sería lematizar el corpus completo de adjetivos. Tras contemplar numerosas opciones, entre ellas usar modelos de OpenAI vía API, se llegó a la conclusión de que la forma más rápida y certera sería usar de nuevo la librería `spaCy`.

Tras finalizar la lematización, se procedió a contabilizar la frecuencia de cada uno de los distintos adjetivos sobre el conjunto completo de características. Al igual que con los sustantivos, habían muchos adjetivos, pero la mayoría no eran relevantes. En consecuencia, se decidió eliminar aquellos que tenían una frecuencia muy baja ya que se observó que no aportaban valor al tener en cuenta todas las reseñas.

Tal como se hizo con los sustantivos, se generó un diccionario de adjetivos con el objetivo de agrupar sinónimos. Por ejemplo, adjetivos como reducido, pequeño, minúscula o baja se unificaron bajo un término representativo: pequeño. A continuación, se reprocesaron todas las características para aplicar estos cambios, sumando las frecuencias de los sinónimos agrupados bajo la nueva características representante.

3.4.9. Resultado final

Al final se obtuvieron un total de veinticinco sustantivos y ciento cincuenta y ocho adjetivos distintos entre todas las reseñas, lo cual supuso una reducción significativa del corpus original. Consecuentemente, como se ve en la figura 3.25, se redujeron significativamente los sustantivos y los adjetivos presentes en la extracción final de las características, manteniendo una gran caracterización del estado de las paradas de metro.

```
Counter({'escalera': Counter({'mecanico': 1, 'ascensor': 1, 'vigilado': 1}),
         'estacion': Counter({'bien': 7,
                  'lleno': 2,
                  'nuevo': 2,
                  'transitado': 1,
                  'ultimo': 1,
                  'limpio': 1}),
         'transporte': Counter({'publico': 1}),
         'cobertura': Counter({'movil': 1}),
         'acceso': Counter({'facil': 1, 'bien': 1}),
         'metro': Counter({'normal': 1})})
```

Figura 3.25: Resultado de aplicar el método de extracción de características sobre Alonso de Mendoza

3.5. Creación de rutas

Llegados a este momento, se disponía de doscientas setenta y cinco estaciones de metro y sus características. El objetivo era representar la red de Metro de Madrid y calcular posibles rutas dadas una estación de origen, otra de destino y una serie de filtros.

3.5.1. Representación de la red de Metro de Madrid

Se consideraron varias alternativas para modelar el entramado del metro. Aunque la más evidente y directa es mediante grafos, se estudiaron otras opciones como matrices de adyacencia, máquinas de estados y una representación mediante estructuras de datos (diccionarios, listas, JSON ...).

Una red de metro puede definirse como un conjunto de puntos (estaciones) conectados por trayectos (líneas). Por tanto, los grafos ofrecen una representación fiel y eficiente.

Escogida la representación, se estudiaron varias librerías para su modelado y análisis: `python igraph` ofrece un envoltorio en C muy eficiente pero una API poco "pythónica". `graph tool`, basado en C++ resulta complejo de instalar, compilar y dispone de una API poco intuitiva. `Networkit` aporta paralelización nativa en C++ para acelerar algoritmos en grandes redes, pero su catálogo de algoritmos es muy limitado. `graphlib`, incluida en la librería estandar de Python, solo permite trabajar con grafos dirigidos no cíclicos. Dado que todo el proyecto se desarrolló en Python, se requería una herramienta que combinase facilidad de instalación, una API "pythónica", extensa documentación y soporte para todos los tipos de grafos. Además, teniendo en cuenta que la red de Metro de Madrid puede considerarse pequeña, `Networkx` se consideró como la opción más adecuada para un desarrollo ágil y totalmente integrado con el resto del proyecto.

Como `NetworkX` permite todo tipo de grafos (simples, multigrafos, grafos dirigidos ...), se estudió cual sería la representación más adecuada. Inicialmente, se optó por un grafo simple en el que las estaciones fuesen los nodos y las conexiones directas entre ellas fuesen las aristas. No obstante, al observar que entre algunas estaciones

existen múltiples conexiones, es decir, de una estación a otra se puede llegar desde dos líneas distintas, fue necesario cambiar la representación a un multigrafo. Aunque esta situación no es común, como puede verse en la figura 3.26, esta situación se da entre las estaciones de Moncloa y Argüelles, a las que se puede llegar mediante las líneas tres y seis así como entre Avenida de América y Diego de León (líneas seis y cuatro).

Figura 3.26: Estaciones paralelas del Metro de Madrid

`NetworkX` acepta como identificador de nodos cualquier objeto "hasheable" de Python. Además, permite añadir cualquier tipo de atributo tanto a los nodos como a las aristas. Consecuentemente, los nodos serían identificados por el id asignado en la base de datos y sus atributos serían, su nombre, su dirección, la descripción, las líneas accesibles desde la estación y el conjunto de características obtenidas en la anterior etapa del proyecto. Por otro lado, las aristas se identifican mediante una tupla cuyo primer elemento corresponde al identificador de la estación de origen y el segundo al identificador de la estación de destino. Además, se añadiría la duración del trayecto de la estación origen a la estación destino como atributo de la arista. Al ser un multigrafo, `NetworkX` permite añadir un valor "key" a las aristas, facilitando la distinción entre múltiples conexiones entre dos nodos. Para este contexto, "key" debía ser la línea.

Con esta primera versión se obtuvo un grafo como el mostrado en la figura 3.27, que representa una pequeña sección del la red de metro.

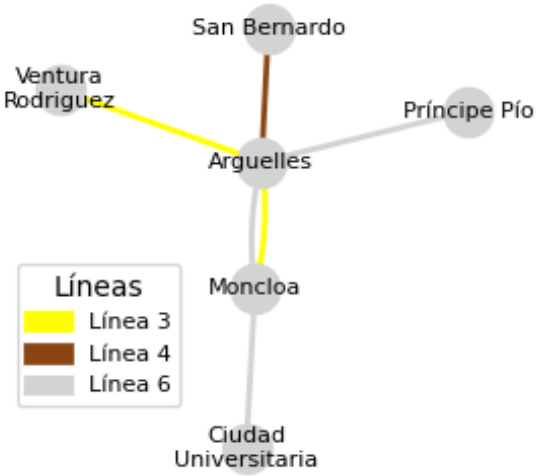

Figura 3.27: Demostración de multigrafo con `NetworkX`

Aunque a nivel conceptual esta representación fuese la más simple y a priori parecía la más acertada, suponía dificultades a la hora del cálculo de rutas óptimas. En

concreto, el mayor impedimento residía en la penalización de los transbordos: en un multigrafo, `Networkx` espera que la función de peso ("weight") dependa únicamente de la arista que se está evaluando. Es decir, no es posible que esta función sepa qué línea se ha utilizado para llegar al nodo origen de la arista en cuestión.

Dado que cualquier penalización por transbordo requiere información histórica que no cabe en el peso de la arista, si se quería seguir con esta representación, habría sido necesario realizar una implementación propia del algoritmo de Dijkstra que hubiese mantenido el estado de la línea anterior para cada nodo en la búsqueda. La función de coste dependería del tiempo del trayecto y de la línea previa.

Realizar esta modificación del algoritmo supondría perder las optimizaciones en C de la versión original. Además, si en un futuro se quisiesen añadir nuevas funcionalidades que afectasen al coste (por ejemplo, ventanas horarias o problemas puntuales en las líneas), sería necesario cambiar la función de coste del algoritmo.

Teniendo esto en cuenta, se decidió cambiar la representación a un grafo simple donde los nodos serían identificados por una tupla cuyo primer elemento sería el id asignado en la base de datos y el segundo elemento sería la línea a la que pertenece la estación. Consecuentemente, aquellas estaciones con más de una línea generarían más de un nodo (Figura 3.28).

Figura 3.28: Representación de Arguelles con grafo simple y nodo con clave id, linea

Las aristas, por tanto, unirían pares de nodos (cada uno definido por una estación y la línea correspondiente) y llevarían como único atributo el tiempo de trayecto, ya que la información de la línea está implícita en el nodo. Su generación se realizó en dos fases: primero, se añadieron las conexiones directas extraídas de la base de datos (enlaces estación a estación por línea). En este punto, como puede verse en la figura 3.29, el grafo queda dividido en tantas componentes conexas como líneas conforman la red de Metro de Madrid.

Durante la segunda fase se incorporaron las aristas de transbordo, uniendo nodos de una misma estación en líneas diferentes y asignándoles la penalización correspondiente al cambio de línea (penalización por transbordo). Consecuentemente, debido a la estructura de la red de Metro de Madrid, el grafo final era conexo.

En la figura 3.30 puede verse la misma sección del metro ilustrada en la figura 3.27 pero con la representación definitiva (nodos identificados por estación y línea).

Figura 3.29: Fase uno de la introducción de aristas (red dividida en componentes conexas)

3.5.2. Cálculo de rutas

Una vez representado el grafo, se abordó el problema del cálculo de rutas óptimas. Para ello, se exploraron distintos algoritmos de búsqueda como A* y Dijkstra. Dado que la librería `Networkx` ya incluye implementaciones eficientes basadas en Dijkstra, se optó por utilizar la función *single_source_dijkstra*. Esta permite obtener tanto el coste mínimo desde un nodo origen como el camino correspondiente, evitando la necesidad de ejecutar múltiples veces el algoritmo.

El principal desafío consistía en encontrar una ruta que no solo fuese óptima en términos de tiempo, sino que también respetara, en la medida de lo posible, los filtros definidos por el usuario. En caso de que no existiera una ruta que cumpliera todos los criterios, era necesario ofrecer la alternativa más adecuada disponible. Para abordar este problema, se optó por modificar el grafo original mediante tres estrategias distintas:

- **Ruta en la que todas las estaciones que componen el trayecto cumplen todos los filtros**: el primer enfoque era encontrar una ruta donde todas las estaciones cumplieran todos los filtros que el usuario estableciese. Para ello, se eliminarían aquellos nodos (estaciones) que no cumpliesen alguno de los filtros introducidos. A continuación, se lanzaría Dijkstra para obtener la mejor ruta.

  Como resultado de esta modificación, en la mayoría de los casos, el grafo resultante se volvía inconexo. Esto se debía a la falta de información disponible, provocada principalmente por la escasez de reseñas, lo que impedía asignar características relevantes a muchas estaciones. En consecuencia, las estaciones disponibles presentaban propiedades muy dispares, y por lo tanto en el proceso de eliminación de nodos se descartaban la mayoría de ellos. Esto dificultaba o

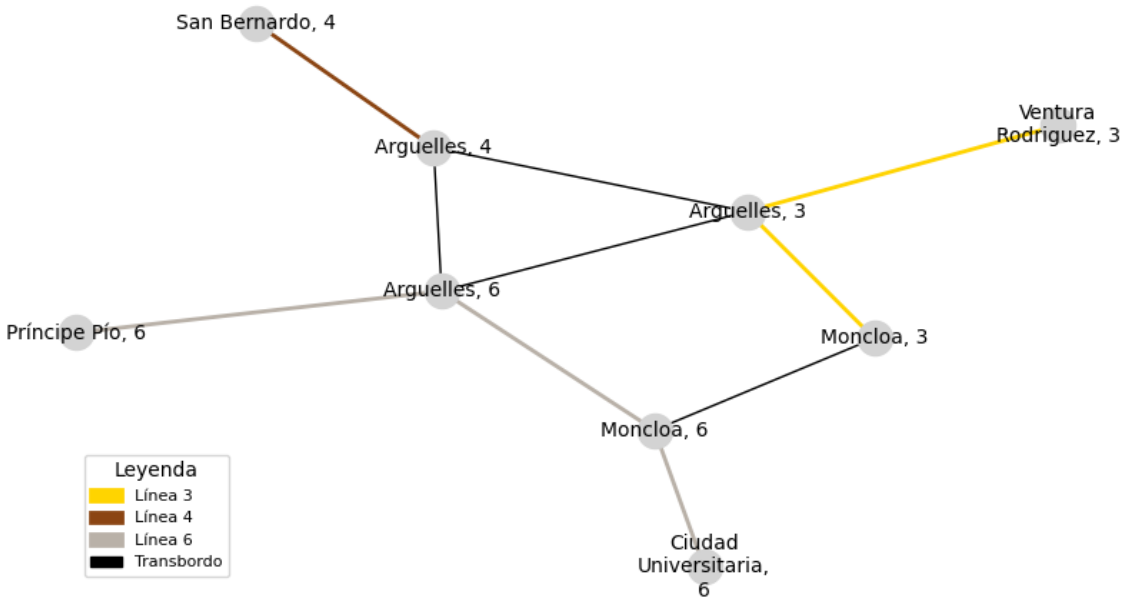

Figura 3.30: Demostración de grafo simple con nodos identificados por la estación y la linea

incluso imposibilitaba, la generación de rutas en las que todas las estaciones cumplieran simultáneamente los filtros establecidos.

- **Ruta en la que todas las estaciones de transbordo que componen el trayecto cumplen todos los filtros**: el segundo enfoque es muy parecido al primero con la diferencia de que al modificar el grafo solo se eliminarían aquellos nodos que fuesen estaciones de transbordo (con más de una linea). Es decir, se forzaría a que las estaciones intermedias en las que el usuario se viese forzado a bajar (estaciones de transbordo) cumpliesen todos los filtros. Sin embargo, las estaciones intermedias (en las que el usuario no se tendría que bajar) no se tienen en cuenta para el cumplimiento de los filtros, y por lo tanto nunca se eliminan.

  Este método, aunque mucho menos restrictivo que el anterior, sigue sin garantizar encontrar ruta.

- **Ruta más rápida cumpliendo si es posible los filtros**: el objetivo principal del tercer enfoque era obtener siempre una ruta: ruta que maximiza el cumplimiento de los filtros y minimiza el tiempo de trayecto.

  En este caso se revaluaría el grafo pero sin eliminar nodos, tan solo se recalcularían los pesos de las aristas añadiendo una penalización en caso de no cumplir con los filtros y un premio en caso contrario. Además, se decidió que solo se alteraría el peso de aquellas aristas que fuesen de transbordo (aristas compuestas por nodos con la misma estación pero distintas lineas). Como en el caso anterior, esto se realizó porque una vez el usuario estuviese montado en el metro, no notaría si las estaciones por las que pasa cumplen o no los filtros a menos de que se tenga que mover en ellas para hacer un cambio de línea.

Como Dijkstra no permite aristas con pesos negativos, las penalizaciones y los premios se calcularían a través de una función multiplicativa: en caso de penalización el multiplicador sería un número entre cero y uno (sin incluir) y en caso contrario sería mayor a uno y menor que dos. El multiplicando siempre sería el tiempo de trayecto (tiempo de desplazamiento de una estación a otra). De esta manera se aseguraría que el peso de la arista nunca fuese menor que cero.

Como no todas las características son igual de importantes (por ejemplo, que una estación no sea grande no es igual de importante que una estación tenga acceso para personas con movilidad reducida), se decidió dividir las características en tres grupos: "hard" para características muy importantes (penalización: 0.66 y premio: 1.5), "medium" para características importantes pero no sumamente prioritarias (penalización: 0.75 y premio: 1.25) y "soft" para las menos relevantes (penalización: 0.88 premio: 1.1). Estos valores se decidieron para premiar más el cumplimiento de los filtros, dado que si dos estaciones son de transbordo y una cumple los filtros y la otra no, el factor multiplicativo para la arista será un número menor o igual a uno.

Al contrario que los otros dos enfoques, este siempre sería capaz de obtener una ruta.

Finalmente, para cada petición del usuario se decidió ejecutar las tres variaciones y presentárselas para su elección.

3.5.3. Creación de nodos virtuales

Al ejecutar los algoritmos de generación de rutas surgió una nueva dificultad relacionada con la representación de los nodos, ya que cada nodo correspondía a una combinación de estación y línea. Esto significaba que, si una estación pertenecía a más de una línea (como ocurre en las estaciones de transbordo), se generaban múltiples nodos para la misma ubicación física. Esto ocasionaba un problema cuando el origen o el destino elegido por el usuario era una estación de este tipo, ya que no era posible determinar de forma directa un único nodo de inicio o llegada.

Para resolver este inconveniente, se optó por introducir nodos virtuales conectados a los nodos de transbordo en los casos en que estos actuaran como origen o destino de la ruta. Estos nodos virtuales no poseen características propias ni están asociados a ninguna línea, y se enlazan a los nodos correspondientes mediante aristas de coste cero. De este modo, la transición desde un nodo virtual a cualquier nodo real asociado se realiza sin penalización, garantizando una correcta ejecución del algoritmo.

Como se puede ver en la figura 3.31, para la sección de la red de metro descrita anteriormente, si Moncloa se escogiese como estación origen (o destino) sería necesario añadir un nodo virtual conectado a los nodos correspondientes a las lineas de Moncloa mediante aristas de coste cero.

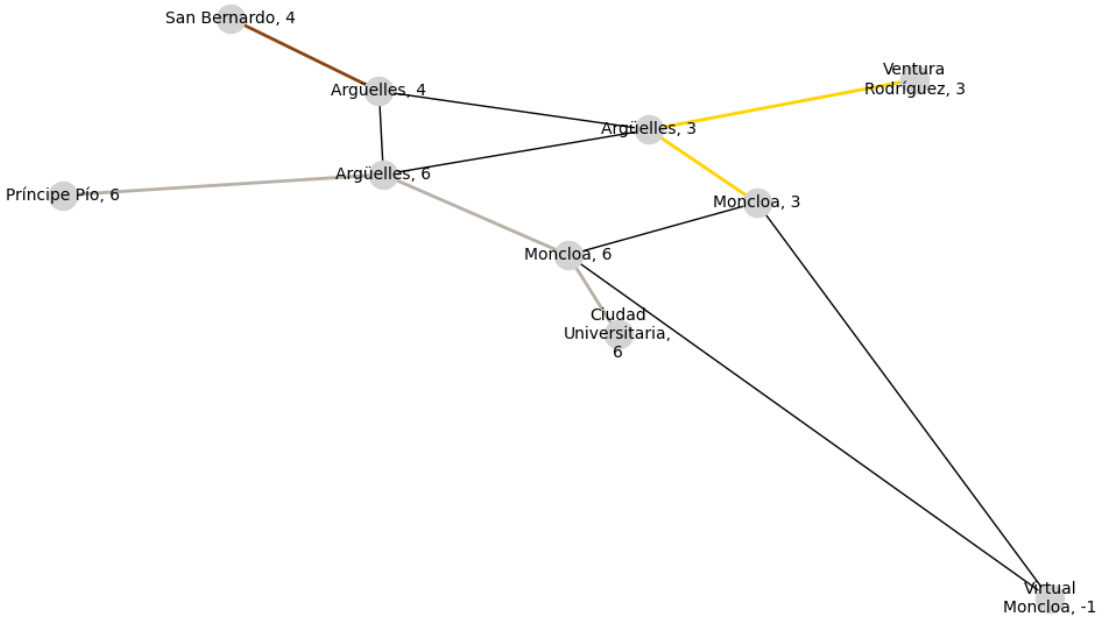

Figura 3.31: Ejemplo de nodo virtual

3.5.4. Resultado final del cálculo de rutas

Al final, como se ve en la figura 3.32, se obtendrían entre una y tres rutas que serían las que se mostrarían a través de la web para que el usuario pudiese elegir la que más le convenga.

3.6. Diseño e implementación de la API

La API constituye la puerta de entrada al motor de rutas del proyecto. Su misión es sencilla: recibir una solicitud en la que se especifica la estación de origen, la de destino y una lista opcional de filtros, comprobar que los datos sean correctos, calcular una serie de posibles trayectos y devolver el resultado en formato JSON.

El servicio se apoya en `FastApi`, una herramienta ligera que encaja bien cuando se busca rapidez y capacidad para atender muchas consultas a la vez. Al arrancar la aplicación se carga en memoria el grafo que representa la red de Metro de Madrid. De este modo, cada petición accede a ese mapa ya disponible y evita la consulta a la base de datos, que sería mucho más lenta.

La razón principal para escoger `FastAPI` radica en su equilibrio entre rendimiento y facilidad de uso. Su diseño asíncrono facilita que varias personas consulten la API al mismo tiempo sin bloquearse unas a otras; la validación automática de los datos evita errores antes de que lleguen al corazón del sistema; y la documentación interactiva que genera el propio framework reduce el esfuerzo de integración con otras aplicaciones.

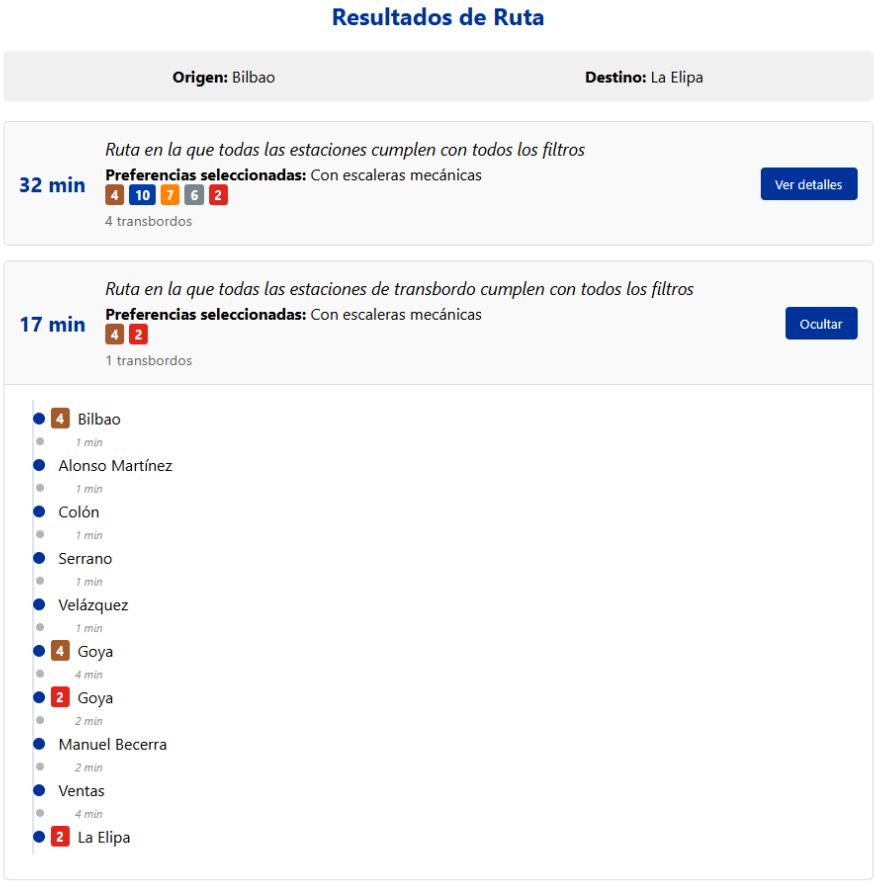

Figura 3.32: Obtención de rutas

El resultado es un servicio rápido, claro y sencillo de mantener, preparado para crecer conforme se amplíen las necesidades del proyecto.

3.7. Estudio y preparación para la creación de la aplicación web

Aquí, se describirá paso a paso los procedimientos que hemos seguido para la realización de la página web del proyecto.

3.7.1. Plataformas de bocetos y diseños de interfaces

En cuanto al proceso que hemos llevado a cabo para realizar el boceto inicial de la web, exploramos y aprendemos las distintas plataformas que existen. Dicho esto, estas son plataformas que ofrecen desde herramientas para UI design, prototipos, hasta colaboración a través de tiempo real.

Algunas de estas alternativas son populares entre diseñadores y equipos de desa-

rrollo, y tienen características propias que pueden ser más adecuadas dependiendo del flujo de trabajo y necesidades del equipo. Ejemplo:

Sketch

`Sketch` es una de las herramientas de diseño más importantes y se da a conocer entre los denominados "Apple Developers" debido a que es algo exclusivo para macOS. `Sketch` se dedica esencialmente a la creación de interfaz de usuario e incluso puede prototipar; incluye una cantidad enorme de *plugins* que hace que su uso sea mucho más versátil. Esta plataforma no es accesible en el navegador y al momento de finalizar la edición permite exportar archivos trabajados y compartir con los demás miembros del equipo. Una debilidad de `Sketch` es que no tiene funciones de colaboración en tiempo real en su versión básica. Pero, al mismo tiempo, es una buena recomendación entre diseñadores que solo trabajan en entornos de Apple y equipos que quieren herramientas de diseño compatibles con versiones anteriores.

Adobe XD

`Adobe XD` es una herramienta robusta para el diseño de interfaz y prototipos, se usa en ambos soportes: macOS y Windows. `Adobe XD` es interactiva y permite realizar grandes diseños en su entorno, ya que se asimila a `Figma` sin el trabajo con vectores, gradientes y tipografías. Su mejor parte es que está diseñada para la colaboración en tiempo real a través de Adobe Creative Cloud, de modo que es mucho más colaborativo, pues uno va compartiendo los enlaces para las reseñas de diseño.

InVision

`InVision` es una herramienta web centrada en prototipado y prueba de usuarios. Está pensada para simplificar la creación de prototipos visuales y el trabajo colaborativo en proyecto de diseño. Aunque es excelente para prototipado, también ofrece colaboración de uso en tiempo real: los equipos pueden dejar comentarios en los diseños directamente. Puede considerarse su mayor ventaja su amplio número de herramientas de pruebas de usuario y de tareas, lo cual hace de ella el elemento ideal para situaciones donde uno tiene que entregar constante *feedback*. Es decir, es interesante para equipos que estén centrados en crear prototipos interactivos y validar en permanente contacto con los usuarios.

Framer

`Framer` es una herramienta de diseño via web aunque tiene versión para macOS; se trata de una herramienta a usar para la creación de prototipos de alta calidad con una excelente cantidad de interacciones, no obstante puede ser limitada para aquellos que quieran avanzar un poco más en la personalización. Una de sus mayores ventajas son las animaciones y transiciones que puedes crear de manera inmediata y

sin mucho esfuerzo, apta para los diseñadores que desean crear potentes prototipos de gran realismo.

Axure RP

`Axure` es una herramienta poderosa destinada a Windows y macOS, dirigida a diseñadores que necesitan crear prototipos de gran cantidad de complejidad. Principalmente, es famosa por tener capacidad para lógica dinámica, incluir bases de datos, formularios y cualquier otra interacción "más técnica" por encima de prototipado estándar. Toda esta gran cantidad de posibilidades técnicas permite diseñar prototipos muy completos que, de forma pragmática, simulan el funcionamiento de productos originales. Por lo tanto, tiene sentido utilizar `Axure` si se "quiere" ir más allá de la simplicidad y requiere proyectos de grandes capacidades.

Canva

`Canva` es una herramienta en la nube que no se centra directamente en el diseño de interfaz como `Figma` o `Sketch`, pero sin duda es extremadamente útil para hacer gráficos, presentaciones, ensamblados de mercadotecnia u otros contenidos. Su esencia es la sencillez de su uso, la interfaz intuitiva hace que incluso un diseñador principiante puede usarla junto con personas sin experiencia. Buena opción para producir material de visualización fácil y gracias a su enfoque práctico y sencillez, `Canva` es especialmente útil para equipos donde no hay profesional de diseños.

Proto.io

`Proto.io` es una herramienta basada en la web diseñada para facilitar la creación de prototipos interactivos de aplicaciones móviles y sitios web. Permite diseñar interfaces y añadir interactividad sin necesidad de escribir código, lo que la convierte en una opción accesible incluso para quienes no tienen conocimientos técnicos. Su interfaz es sencilla e intuitiva, y está especialmente optimizada para el desarrollo de prototipos móviles, lo que la hace ideal para equipos que trabajan principalmente en el diseño y prueba de aplicaciones móviles.

Balsamiq

`Balsamiq` es una opción no solo web, sino para macOS y Windows, y está diseñada exclusivamente para crear *wireframes* rápidamente y de manera sencilla. `Balsamiq` es una herramienta enfocada en la creación de *wireframes* de baja fidelidad, en contraste con otras plataformas de diseño centradas primordialmente en la alta fidelidad. La simplicidad es una de las mayores características porque permite a los diseñadores captar la idea y estructura de manera rápida y fluida sin tener que abordar detalles de diseño más altos. Es particularmente útil para quienes necesitan crear rápidamente un prototipo básico o, en general, para aquellos equipos que inician el proceso de definir la construcción de sus productos.

Gravit Designer

`Gravit Designer` es una herramienta basada en la web que está distribuida en macOS, Windows y Linux. Es una opción accesible desde prácticamente cualquier sistema operativo ya que, a nivel de competencias técnicas, se encuentra por debajo de las otras herramientas mencionadas. Se centra en el hecho de ofrecer una solución versátil para la creación de GUI, ilustraciones, modelos gráficos o baja fidelidad. Ofrece una opción de pago con un precio bajo que hace que sea una opción interesante para aquellos que no tengan un presupuesto grande a la hora de realizar un proyecto.

Plataforma elegida: Figma

`Figma` es una herramienta de diseño basada en la web conocida por sus características colaborativas que permiten a diferentes usuarios trabajar en el mismo archivo al mismo tiempo y ver actualizaciones de forma instantánea. Esto mejora enormemente la coordinación entre los miembros del equipo, incluso si están ubicados en diferentes regiones. Además, los usuarios pueden comentar directamente sobre elementos de diseño específicos, lo que agiliza la retroalimentación y reduce reuniones o correos electrónicos innecesarios. Una de las mejores características de `Figma` es la capacidad de crear componentes re-utilizables, como botones y formularios, que se actualizarán automáticamente a lo largo del proyecto cuando se hagan cambios. Aunque `Figma` puede carecer de todas las opciones más avanzadas que se encuentran en otras plataformas, proporciona una solución adaptable y sencilla en forma de una interfaz bien estructurada que puede ser navegada tanto por profesionales como por principiantes.

- **Colaboración en tiempo real**: Mejora la productividad al permitir que la comunicación en tiempo real entre diversos usuarios trabajando en el mismo archivo.
- **Trabajo en la nube**: Puede realizarse desde cualquier lado sin la necesidad de un software adicional.
- **Prototipos interactivos**: Se pueden crear prototipos sin necesidad de ingresar ninguna línea de código.
- **Control de versiones y seguridad**: Los registros de cambios permiten tener control sobre las versiones que se restauren.
- **Sincronización automática**: Todos los integrantes del equipo tienen acceso en tiempo real a los datos que se vayan cambiando o actualizando.
- **Equipos de diseño y desarrollo**: `Figma` es perfecto para equipos que están en la fase de diseño y desarrollo de apps, webs o otros productos digitales ya que les permite ser altamente funcionales.

Figura 3.33: Página de bienvenida de nuestra web

- **Diseñadores UI/UX**: Es ideal para los profesionales que trabajan en el diseño de las interfaces y experiencias de los usuarios debido a su gran capacidad.

- **Equipos distribuidos**: `Figma` es una gran opción para los equipos que están geográficamente distribuidos y trabajan a distancia por la web, debido a su interfaz con base en la nube.

3.7.2. Diseño del boceto en Figma

Antes de comenzar el prototipo del sitio web utilizando `Figma`, nos aseguramos de dedicar suficiente tiempo a explorar las herramientas para utilizarlas de manera óptima, revisando varios tutoriales y videos en internet para familiarizarnos con la plataforma, sus herramientas y sus funciones avanzadas. De esta manera, construimos una base sólida y evitamos rehacer trabajos innecesarios, optimizando nuestro flujo de trabajo.

Junto con la teoría, fuimos practicando con algunas habilidades esenciales como crear marcos, capas y trabajar con componentes re-utilizables. Utilizar estas herramientas en la etapa inicial nos ayudó a generar confianza, agilizar el proceso de diseño y familiarizarnos con la interfaz y los componentes, a la vez que descubrimos trucos ocultos adicionales.

Una vez que nos sentimos cómodos con Figma, comenzamos a trabajar en el prototipo del sitio web prestando atención a adaptar el diseño junto con nuestros objetivos individuales, aprovechando el aprendizaje colectivo de los pasos anteriores. El enfoque secuencial aseguró que abordáramos el diseño de una manera más organizada y exhaustiva, evitando bucles en las herramientas, que pueden descarrilar los flujos de trabajo, y en su lugar, enfocándonos en los detalles y soluciones creativas.

3.7.2.1. Estructura y objetivo de la web

`Figma`, cómo cualquier aplicación web de estas características, tiene una versión de pago que te permite profundizar y usar la aplicación de una forma más completa. En nuestro caso no ha hecho falta usar la versión de pago ya que la versión gratuita nos ha permitido realizar al menos una pequeña base de cómo sería nuestra web.

Cómo decimos, la versión gratuita de `Figma` tiene limitaciones y una de ellas era que únicamente nos ha permitido la creación de un máximo de 4 páginas distintas. Debido a ello, hemos tenido que reciclar alguna de las páginas para poder bocetar todas las páginas de nuestro proyecto.

Nuestra intención era realizar una aplicación web sencilla en la que los usuarios puedan interactuar de una forma fácil e intuitiva. La web está estructurada en diferentes secciones. En la parte superior, encontramos el encabezado de la web en un tono azul intenso, en la izquierda podemos ver el logo del metro de Madrid, en el centro el título "Good Stations" en negrita, que destaca el título de la web, y en la esquina derecha, se ha añadido el escudo de la Comunidad de Madrid para reforzar la identidad local del servicio (véase figura 3.33). En el centro de la pantalla, el logo del Metro de Madrid actúa como el elemento visual principal, captando la atención del usuario y dejando claro el contexto de la aplicación. El fondo en un tono blanco aporta un diseño limpio y moderno. En el extremo inferior, se encuentra un botón de accesos con la leyenda "ENTRAR" para llevar al usuario al siguiente paso de la plataforma que se le presente. Se destaca por el color azulado sobre fondo, para mayor facilidad de interacción.

El dibujo pretende dar claridad y simplicidad a la interfaz, para que el usuario pueda navegar intuitivamente desde el primer momento que entre en la página.

Una vez pulsamos el botón "ENTRAR" en la pantalla de bienvenida de la misma, llegamos a la página principal de Good Stations. En dicha página, los usuarios pueden introducir el trayecto que desean realizar en el Metro de Madrid según filtros y opiniones de otros usuarios, cuyas valoraciones son conocidas en el programa.

El diseño está dividido en varias secciones. En la parte superior, se mantiene la identidad visual con el título `Good Stations`, el logo del Metro de Madrid a la izquierda y el escudo de la Comunidad de Madrid en la derecha. Además, se ha añadido un menú de navegación en la esquina superior derecha con las opciones `Inicio`, `Estaciones` y `Acerca`, facilitando el acceso a otras secciones de la web que detallaremos a continuación. En el cuerpo principal, se encuentra el formulario de búsqueda de trayectos, compuesto por dos campos donde el usuario puede ingresar su estación de origen y destino. A la derecha de estos campos, hay un botón de filtrado, que permitirá personalizar la búsqueda según criterios como seguridad, limpieza, comodidad, etc. Finalmente, en la parte inferior, encontramos el botón Buscar que tras pulsarlo inicia la búsqueda del trayecto y nos lleva a la pantalla que detallaremos seguidamente.

Tras pulsar el botón Buscar en la pantalla anterior, llegamos a esta nueva página donde se muestra el detalle del trayecto optimizado según las valoraciones de los usuarios. La estructura general se mantiene igual, con el título `Good Stations`, el

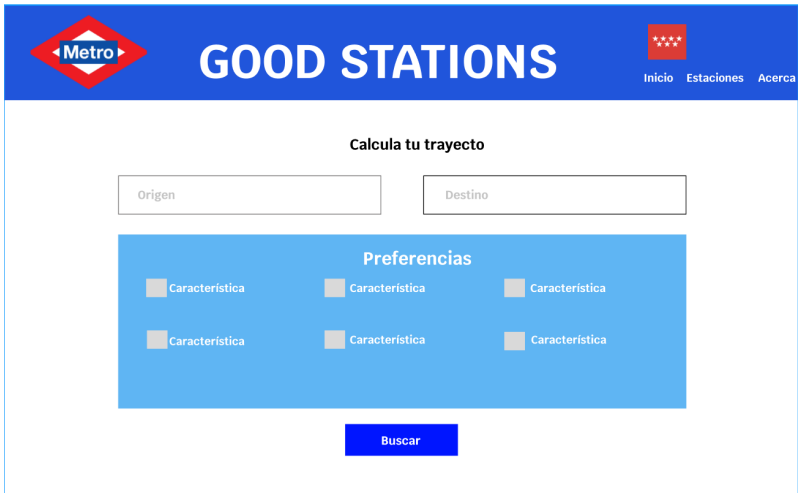

Figura 3.34: Página de Inicio y búsqueda de trayecto

Figura 3.35: Página del trayecto descrito según los criterios seleccionados

logo del Metro de Madrid a la izquierda, el escudo de la Comunidad en el centro y el menú de navegación en la parte superior derecha. Véase figura 3.35.

La diferencia principal está en la zona central, donde antes estaba el formulario de búsqueda. Ahora, en su lugar, se muestra el trayecto detallado, con la estación de origen, la de destino y las paradas intermedias recomendadas según los criterios seleccionados. También se destacan aspectos como la valoración de cada estación, posibles incidencias mencionadas en las opiniones y opciones para modificar la ruta si el usuario desea ajustarla.

Vamos a explicar lo que hacen los tres botones del menú principal:

- `Inicio`: Al hacer clic en este botón, el usuario volverá a la pantalla de bienvenida de la web, dónde deberá pulsar de nuevo en el botón `Entrar` que le llevará a la página de elegir trayecto donde podrá ingresar nuevas estaciones de origen y destino para calcular una ruta optimizada.

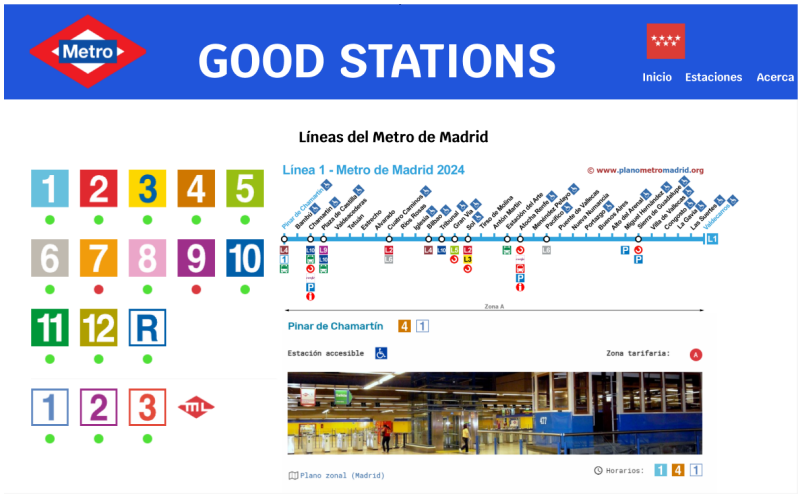

Figura 3.36: Página de las líneas y estaciones del Metro de Madrid

- `Estaciones`: Esta opción lleva a una página que muestra un listado de todas las líneas y estaciones del Metro de Madrid. Cada línea desplegará las estaciones que tiene y en cada estación veremos que incluye una pequeña descripción de la misma.
- `Acerca`: En esta sección, el usuario encontrará una breve descripción del proyecto Good Stations, explicando su propósito y por quién está realizado.

Siguiendo con la descripción de la web, entramos en la sección de "Estaciones". En la parte superior, se mantiene la identidad visual con el título "Good Stations", el logo del Metro de Madrid a la izquierda y el escudo de la Comunidad de Madrid en la derecha junto con el menú de navegación en la esquina superior derecha con las opciones "Inicio", "Estaciones" y "Acerca", facilitando el acceso a otras secciones de la web que detallaremos a continuación. En la parte central, se muestran todas las líneas del metro identificadas por sus colores oficiales y números (de la 1 a la 12, más las líneas Ramal y ML1, ML2, ML3). Cada línea está acompañada por un punto verde o rojo, pero ésto forma parte del boceto, en nuestro proyecto final no se mostrará dicho punto. Cada línea desplegará las estaciones que tiene y en cada estación veremos una pequeña descripción de la misma. En nuestro caso hemos puesto cómo ejemplo el plano de la Línea 1, con todas sus estaciones y los símbolos que indican conexiones con otras líneas, ascensores, aparcamientos o puntos de información. La estación que se destaca en este ejemplo es Pinar de Chamartín. Hemos incluido una foto real de la estación, los horarios según cada línea y la accesibilidad. Véase figura 3.36.

Por último, hemos añadido la sección "Acerca" de la web Good Stations. Aquí se explica brevemente el objetivo principal del trabajo: crear una aplicación web que permita personalizar rutas dentro del metro basándose en valoraciones y experiencias reales de los usuarios. La idea es que cada persona pueda encontrar la mejor ruta no solo por rapidez, sino también por comodidad, accesibilidad o preferencia personal. Debajo del texto principal hemos añadido un apartado donde se mencionan las tecnologías que se han utilizado para desarrollar el proyecto (en este caso aparecen como ejemplos: Tecnología 1, 2 y 3). Este espacio se detallarán si se ha usado HTML,

CSS, JavaScript, alguna librería específica, etc. Finalmente, se muestra una lista con los miembros del equipo que han trabajado en el desarrollo del proyecto así cómo una pequeña descripción de los mismos. Al igual que se mencionará a los profesores y responsables del proyecto. Véase Figura 3.37.

Figura 3.37: Página de la sección "Acerca" del proyecto

3.7.3. Evaluación del prototipo con usuarios

Se va a realizar una pequeña prueba sobre distintos usuarios para la correcta evaluación del prototipado.

3.7.3.1. Propósito y objetivos de la evaluación

El propósito de realizar una evaluación es valorar la interactividad, accesibilidad y usabilidad de la aplicación para ofrecer la mejor experiencia posible a los usuarios.

Los objetivos de ésta son los siguientes:

- Verificar que los usuarios sean capaces de encontrar los controles necesarios para navegar dentro de la aplicación.
- Anotar los errores que cometen los usuarios para mejorar la interfaz.
- Comprobar que la navegación sea intuitiva.

Se realizará una evaluación con usuarios utilizando un prototipo creado en `Figma`, ya que es la forma más directa de ver cómo va a interactuar el usuario final con nuestra aplicación.

3.7.3.2. Preguntas de investigación

Se emplearán una serie de preguntas para monitorizar la experiencia del usuario y la funcionalidad de la aplicación:

- ¿Los usuarios pueden navegar de forma fácil por la plataforma?
- ¿Los usuarios saben qué elementos son accionables y cuáles no?
- ¿Los usuarios tienen suficiente control y libertad?
- ¿Se pueden realizar las acciones importantes con pocas pulsaciones?
- ¿El usuario es capaz de interactuar con la aplicación de manera correcta?
- ¿Los flujos de la aplicación son fáciles de entender?

3.7.3.3. Descripción del diseño experimental

Introducción al prototipo y contexto

Se va a presentar un prototipo de nuestra aplicación. El objetivo es evaluar el diseño actual con usuarios reales para obtener *feedback* e información sobre nuestra app. Las entrevistas se realizarán de manera presencial u online en función de la disponibilidad de los entrevistadores y los entrevistados.

Listas de tareas a realizar

- Entrar a la aplicación
  - Usar el botón de la pantalla de inicio
- Planificar una ruta
  - Seleccionar una estación de origen
  - Seleccionar una estación de destino
  - Seleccionar una de las opciones extras
- Observar las recomendaciones
- Menú de navegación
  - Usar el menú de navegación para ver las diferentes páginas que contiene la web
- Estaciones
  - Ir usando el menú a la página de Estaciones
  - Seleccionar una estación cualquiera y mirar su información
  - Mirar una estación de una línea distinta para ver la navegación entre líneas
- Acerca
  - Ir a esta página para leer información sobre el proyecto contenida en la web

Descripción del entorno y herramientas a utilizar

Las sesiones se realizarán de manera presencial u online, preferentemente en espacios tranquilos para que los usuarios realicen las tareas sin interferencias externas. Durante la entrevista habrá un moderador presente.

El usuario contará con un ordenador portátil en el que estará instalado el prototipo de la aplicación. Además, se empleará un dispositivo móvil para grabar tanto al usuario como al entrevistador. Adicionalmente, se usará un programa de captura de pantalla para monitorizar las acciones realizadas por el usuario. En el caso de las entrevistas online, este programa también se usará para grabar al usuario y al moderador.

Tareas del entrevistador

El entrevistador tendrá que hacer que el usuario use la técnica de think-aloud, que consiste en que el usuario explique en voz alta lo que está pensando al realizar la tarea.

Además, el entrevistador estará al lado del usuario dándole las tareas que tiene que realizar y ayudarle a completarlas en caso de que tarde demasiado o se agobie

Descripción de los datos a recopilar

Se recogerán los siguientes datos:

- **Comentarios negativos:** Durante el proceso de *Think-aloud* se apuntará cualquier comentario que no sea favorable hacia el diseño o su funcionalidad, considerando que pueda ser una crítica constructiva para mejorar la aplicación.
- **Comentarios positivos:** Durante el proceso de *Think-aloud* se apuntará cualquier comentario positivo o acierto hacia el diseño o su funcionalidad.
- **Opinión sobre la interacción con la plataforma:** En caso de que el usuario gaste demasiado tiempo buscando una funcionalidad, podemos preguntar si esperaba un diseño diferente o cuál es la principal razón de la confusión.
- **Sugerencias de usuarios:** Al final de la entrevista se apuntarán las posibles sugerencias que puedan tener los usuarios para contribuir a futuras mejoras del diseño de la aplicación.
- **Tasa de errores:** Durante el proceso de evaluación se apuntarán los errores relevantes cometidos por el usuario. No se tienen en cuenta errores producidos por las limitaciones técnicas de Figma.

3.7.3.4. Preparación del entorno de evaluación

El participante dispondrá de un ordenador portátil donde estará instalado el prototipo de la aplicación. También se utilizará un teléfono móvil para registrar en vídeo tanto al participante como a la persona encargada de la entrevista. Además, se empleará un software de grabación de pantalla para observar las acciones llevadas a cabo por el usuario. En el caso de que la entrevista se realice de forma remota, dicho programa servirá igualmente para grabar al usuario y al moderador.

El moderador será el encargado de guiar al usuario durante la entrevista, proporcionándole las instrucciones y las tareas de forma oral, no se mostrará al usuario la lista de tareas.

3.7.3.5. Selección de participantes

La selección de los participantes ha sido rápida y sencilla, hemos recurrido a familiares y amigos. Esto facilita realizar la evaluación en un entorno cercano con complicidad y trato agradable. Además, hemos decidido incluir a perfiles bastante diferentes, desde un perfil más adulto hasta un perfil más joven. A continuación se describe a cada uno de los usuarios seleccionados:

Ana Gallego

Ana Gallego Riaño es una enfermera de 31 años que actualmente se encuentra desarrollando su labor profesional en el Centro Quirón de Talavera de la Reina.

Hace años, residió en Madrid dónde estuvo trabajando en una residencia de ancianos. Es por eso que el consumo de transporte público en su caso ha sido muy elevado, con un uso prácticamente diario del metro de Madrid. Su perfil es ideal para realizar el análisis de nuestra aplicación y así ver que errores demanda y que es mejor para el usuario final.

Alejandro Martín

Alejandro Martín Dasilva tiene 28 años y actualmente se encuentra de guardia de seguridad en el parque temático Puy Du Fou en Toledo. Se define cómo un chico muy sociable, amante del deporte y de los videojuegos. Anteriormente, ha trabajado en varios puestos de trabajo en Madrid dónde ha tenido que hacer un uso elevado del transporte público.

Fan Ye

Fan Ye es un graduado en Ingeniería del Software que tiene 23 años y ha sido estudiante de la facultad. Actualmente se encuentra trabajando en una empresa de desarrollo del software. Hemos seleccionado este perfil de participante ya que al

haber sido estudiante de la facultad nos puede ayudar y dar pautas de qué aspectos de la web le parecen que están bien y cuáles no son lo suficientemente intuitivos.

3.7.3.6. Sesión de grabación

Cuestionario previo

Antes de proceder a la sesión de evaluación, presentamos una serie de preguntas que se harán con el fin de filtrar a los usuarios según su nivel de experiencia e interés en el uso del transporte público.

- ¿Cuántos años tiene?
- ¿Cuál es su ocupación o profesión?
- ¿Se desenvuelve bien con la tecnología?
- ¿Suele viajar mucho en trasporte público?
- ¿Qué tipo de transporte público utiliza?
- ¿Con que frecuencia utiliza el metro?
- ¿Suele buscar estaciones con buenas valoraciones y opiniones de los usuarios?

Cuestionario final

Las preguntas que se presentan a continuación están en consonancia con las preguntas de investigación planteadas previamente en la sección 3.5.3.2.

- ¿Has podido navegar de forma fácil por la plataforma?
- ¿Has sabido qué elementos son accionables y cuáles no?
- ¿Has tenido suficiente control y libertad en la aplicación?
- ¿Has podido realizar las acciones importantes con pocas pulsaciones?
- ¿Ha sido capaz de interactuar con la aplicación de manera correcta?
- ¿Los flujos de la aplicación son fáciles de entender?

3.7.3.7. Evaluación tras las sesiones de evaluación

A continuación presentamos las reflexiones finales relativas a las distintas evaluaciones llevadas a cabo.

Ana Gallego

La entrevista realizada a Ana ha resultado bastante útil respecto al uso de la aplicación web. La usuaria indica que la web le ha parecido intuitiva y fácil de manejar e indica que el diseño está bien pensado para el usuario y que no requiere conocimientos técnicos avanzados para su uso.

Además Ana comenta que no ha encontrado fallos mientras utilizaba la aplicación, lo que da una buena señal sobre su estabilidad y funcionalidad.

Desde el perfil de enfermera de la usuaria le hace especialmente ilusión la opción que muestra si las estaciones tienen accesibilidad para personas con discapacidad. Considera que esta función está muy bien integrada y la destaca como algo muy positivo.

En conclusión, la experiencia de la usuaria refleja una impresión general satisfactoria sobre la aplicación.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

Alejandro Martín

Al igual que Ana, Alejandro Martín también comparte una valoración muy positiva respecto a la web. Según indica, destaca especialmente la facilidad de uso y acceso a la plataforma

Uno de los puntos que más destaca es que ha podido identificar claramente qué elementos son accionables (como botones, enlaces o menús) y cuáles no, lo que demuestra un buen uso de los principios de diseño de interfaz.

Además, indica que ha comprendido sin dificultad los flujos de uso de la aplicación, es decir, ha sabido cómo moverse entre pantallas, buscar información o completar tareas sin necesidad de instrucciones externas. Esto evidencia que la estructura general de la aplicación y su lógica de funcionamiento están bien diseñadas, facilitando una experiencia de usuario fluida.

En conjunto, la opinión del usuario refuerza la idea de que la aplicación web del Metro de Madrid logra ser accesible, clara y funcional, ayudando a los usuarios a orientarse y utilizar sus funciones de manera intuitiva y eficaz.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

Fan Ye

La entrevista con Fan Ye fue bastante importante para nosotros, ya que se trata de un graduado en ingeniería del software, lo que permite observar la aplicación web del Metro de Madrid con un enfoque más técnico y crítico.

Desde su experiencia como usuario, señala errores más técnicos como el posicionamiento de algunos componentes o el tamaño adecuado que éstos deberían tener.

Por otro lado, señala que la aplicación presenta una estructura clara y bien organizada y tanto el acceso como la navegación es sencillo entre sus distintas secciones.

En definitiva, la opinión del usuario aporta una validación más especializada sobre la calidad de la aplicación. Su análisis sugiere que la herramienta no solo es funcional desde el punto de vista del usuario general, sino que también cumple con buenas prácticas de diseño y desarrollo de software.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

3.8. Base de datos

Para poder almacenar y tratar toda la información que hemos ido recopilando se ha procedido a diseñar una base de datos con las características necesarias para su correcto tratamiento.

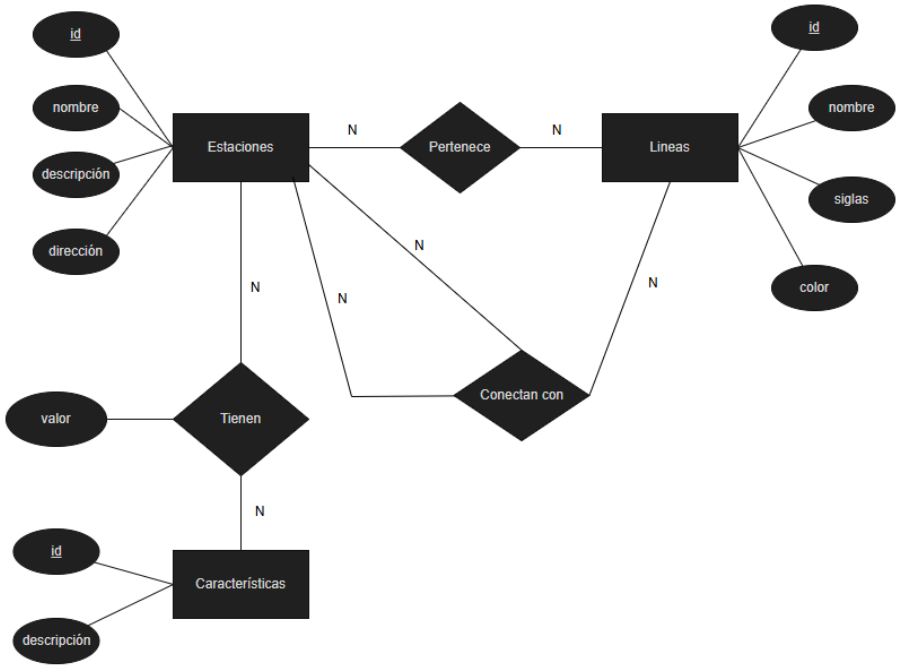

Figura 3.38: Entidad-Relación de la Base de Datos

3.8.1. Diseño de la base de datos

La base de datos está compuesta por diferentes tablas con sus respectivas columnas para almacenar diferentes tipos de datos. Estas tablas se relacionan entre sí a través de distintas claves externas. Para poder explicar estas relaciones se ha procedido a realizar un modelo entidad-relación. Véase figura 3.38.

3.8.2. Estructura de la base de datos

Teniendo el modelo realizado, procedemos a implementar las tablas a la base de datos. Hemos elegido utilizar SQL (Structured Query Language) para ello. A continuación, procedemos a explicar las tablas. Véase figura 3.39.

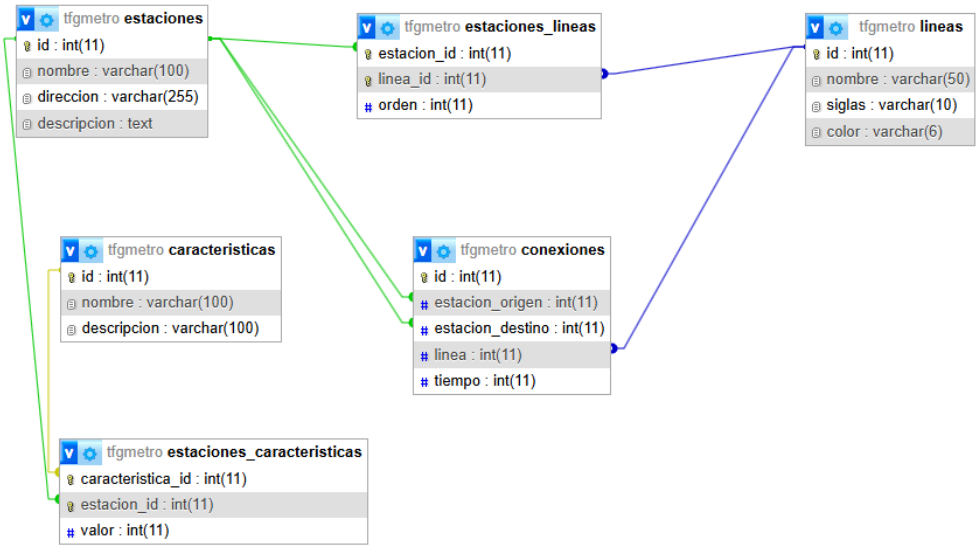

Figura 3.39: Estructura de tablas y campos de la Base de Datos

Estaciones

En esta tabla se almacena la información sobre cada una de las estaciones del Metro de Madrid. Se almacenan 5 atributos:

- id: Identificador de la estación.
- nombre: Nombre de la estación.
- dirección: Dirección donde se encuentra una salida de la estación.
- descripción: Descripción detallada sobre la estación.

Lineas

Esta tabla contiene información sobre las 12 líneas del Metro, el Ramal y las 3 líneas de Metro Ligero. Sus atributos son:

- id: Identificador de la línea.
- nombre: Nombre de la línea.

- siglas: Siglas de los nombres de las líneas (e.g. ML1 para el Metro Ligero 1)
- color: Color de la línea.

Conexiones

Esta tabla representa las distintas aristas que conforman el grafo que forman las estaciones del metro y cómo se relacionan entre sí. Contiene los siguientes atributos:

- id: Identificador de la conexión.
- estacion_origen: Identificador de la estación de origen.
- estacion_destino: Identificador de la estación de destino.
- linea: Línea por la que se produce la conexión.
- tiempo: Tiempo que se tarda en viajar entre las estaciones.

Características

Para poder diseñar la ruta de acuerdo a las preferencias del usuario, se guardan las diferentes propiedades que contiene cada estación. En total se han definido 11 características distintas. Los atributos de la tabla son:

- id: Identificador de la característica.
- nombre: Nombre de la característica.
- descripción: Descripción detallada de la característica.

Estaciones_Características

En esta tabla se asigna a cada estación sus diferentes características. A cada característica se le asigna un valor entre 2 posibles: 1 si contiene esa característica y -1 si contiene el antónimo de esa característica. Los atributos de la tabla son los siguientes:

- característica_id: Identificador de la característica.
- estacion_id: Identificador de la estación.
- valor: Valor que recibe una característica en una estación determinada.

Estaciones_Lineas

Esta tabla muestra la posición en la que se encuentran las estaciones dentro de las líneas.

Los atributos de la tabla son:

- estacion_id: id: Identificador de la estación.
- linea_id: Identificador de la línea.
- orden: Posición que ocupa la estación dentro de la línea (en un sentido).

3.8.3. Implementación de la base de datos

Una vez definida la estructura, procedemos a introducir el contenido de la base de datos. Para ello, hemos extraído los datos de la web oficial del Metro de Madrid. A continuación, hemos adaptado esos datos a la estructura de la web, realizando acciones como asignar identificadores numéricos a los datos que lo necesiten (estaciones, líneas, etc.) e introducir esos datos en las distintas tablas. *Nota: Durante la implementación de los datos, decidimos juntar la estación de Acacias y Embajadores, ya que a efectos prácticos, se trata de la misma estación.*

3.9. Diseño y desarrollo de la web

El desarrollo de la web ha tenido como objetivo principal poner en práctica todo lo aprendido en las distintas asignaturas de programación web. De esta manera, hemos procedido a diseñar y estructurar el código de una manera esquematizada y legible. Para lograrlo, se ha aplicado una esquematización lógica del código, encapsulación de las distintas funcionalidades de la aplicación en clases y una clara división entre el código orientado al aspecto gráfico (CSS) como al aspecto más funcional (php, javascript, etc.).

Cómo decimos, la aplicación ha sido desarrollada utilizando principalmente PHP, HTML, CSS y algo de JavaScript. PHP ha sido el lenguaje base con el que se ha construido toda la lógica del proyecto. Gracias a este lenguaje, hemos podido generar contenido dinámico en función de las peticiones del usuario, gestionar los formularios y conectar con la base de datos. Con HTML se ha implementado la parte estática de la web, permitiéndonos implementar los elementos que conforman a ésta.

Para la parte más estética, hemos utilizado CSS. Nos hemos centrado en crear un diseño fácil y funcional, definiendo los estilos en un único archivo central *estilos.css*. En este fichero hemos incluído el diseño de los distintos elementos, los distintos colores, botones y otros detalles que mejoran la experiencia visual de la web.

Por último, hemos incorporado algo de JavaScript, sobre todo para mejorar la usabilidad de los formularios. Aunque no es el lenguaje principal del proyecto, nos

ha servido para añadir ciertas interacciones que hacen más fluida e intuitiva la experiencia del usuario. Ejemplo de ello es el fichero buscaRuta.js dentro de la carpeta js.

3.9.1. Diseño de la web: versión inicial

El proyecto ha sido organizado en un conjunto de carpetas lo que hace que la estructura sea más clara y lógica.

En la raíz del proyecto se encuentran los archivos principales como index.php, buscarruta.php, resultadosruta.php, lineasmetro.php y acerca.php. Cada uno de estos ficheros representa una funcionalidad concreta del proyecto, y actúan como base de interacción de la web. Los ficheros tienen poco código ya que en estos hemos llamado a los diferentes constructores y a las funciones correspondientes, las cuáles se encuentran en otros ficheros del proyecto. Vamos a proceder a explicar los ficheros principales de la raíz:

- *index.php*: es la página de inicio, donde se muestra el logo de la comunidad de Madrid y el botón de entrada a la aplicación.
- *buscarruta.php*: contiene el formulario para seleccionar el origen y destino del trayecto.
- *resultadosruta.php*: muestra los resultados una vez que el formulario ha sido procesado.
- *lineasmetro.php*: ofrece información detallada de cada línea de metro, y se apoya en la clase Lineas para cargar los datos.
- *acerca.php*: está destinado a explicar el propósito general del proyecto, sus objetivos y los integrantes del mismo

Mostramos un pequeño esquema de la estructura de estos ficheros:

<?php

namespace tfg

require once DIR

tituloPagina

contenidoPrincipal

require DIR plantillas

?>

Figura 3.40: Esquema general ficheros principales

Otras de las carpetas clave del proyecto es */includes*, que reúne toda la parte común y reutilizable del código. Dentro de ella se encuentra el archivo *config.php*, que centraliza la configuración global del sitio, como rutas y constantes. También se incluye un subdirectorio */db*, donde se encuentran ficheros SQL necesarios para crear y alimentar la base de datos del proyecto cómo son los ficheros *tfgmetro.sql* que almacena el esquema de las tablas y campos, y *metroDatos.sql* que almacena los datos de las diferentes tablas.

La carpeta */includes/src* es donde se concentra la lógica más importante de la aplicación. Aquí se encuentran varias clases que encapsulan distintas funcionalidades. Por ejemplo, *Aplicacion.php* actúa como el núcleo que gestiona las conexiones con la BBDD comprobando que se encuentra inicializada y funcional. Luego tenemos otro fichero como *Formulario.php*, que funciona como una base abstracta para la creación de los distintos formularios. A raíz de esta clase, hemos desarrollado los ficheros *FormularioElegirRuta.php* y *FormularioResultadosRuta.php*, encargadas respectivamente de mostrar el formulario de selección de trayecto y de procesar los resultados. Igualmente hemos incluido ficheros de clases como *Lineas.php*, que gestiona y exporta toda la información de las líneas de metro con sus respectivas estaciones.

Para la parte estética común, tenemos una carpeta */vistas* dividida en dos subcarpetas: */vistas/comun* que contiene dos ficheros (*cabecera.php*, *cabeceraIndex.php*), ambos ficheros contienen el diseño de la cabecera del proyecto, y otra subcarpeta */vistas/plantillas* con dos ficheros dos ficheros (*plantilla.php*, *plantillaIndex.php*) que contienen el diseño de la plantilla general de la web. Esto permite mantener una coherencia visual en todas las páginas, sin tener que repetir código. El diseño gráfico se encuentra en la carpeta */css*, donde he centralizado todos los estilos de todas las páginas de la web en un solo archivo, *estilos.css*. Esta hoja de estilo define la apariencia general del sitio: colores, disposición de los elementos, tipografías, etc.

Para finalizar hemos incluido un pequeño script en JavaScript, *buscaRuta.js*, dentro de la carpeta */js*, que permite validar ciertos datos a la hora de elegir estaciones sacando información dinámica sobre la web.

En cuanto a como está diseñado el proyecto, hemos seguido un enfoque orientado a objetos en PHP. Cómo ya hemos comentado, esto nos ha permitido encapsular la lógica en clases independientes, aprovechando principios como la herencia o el polimorfismo. Un buen ejemplo es la clase Formulario, que actúa como base para todos los formularios y permite crear nuevas versiones manteniendo una estructura común. Esta forma de trabajar facilita muchísimo la reutilización del código y mejora la claridad del mismo.

3.9.2. Diseño de la web: versión definitiva

Hemos añadido dos secciones diferentes en la memoria, una con la versión inicial de la estructura del proyecto web (sección anterior) y una versión definitiva con todos los cambios que hemos realizado en el diseño y la funcionalidad de la web (sección actual).

Inicialmente, la estructura de carpetas del proyecto es la misma, mantenimiento el mismo esquema descrito en la sección anterior. La diferencia principal de la carpeta raíz es que hemos eliminado el fichero *index.php* descrito por el fichero *buscarruta.php* que pasa a ser el nuevo *index.php*, esto es debido a que analizando la totalidad de la web hemos visto que ese *index.php* con el simbolo del Metro y el botón `Entrar` no era visiblemente atractivo y funcional para una versión definitiva de la web. Por ese motivo, hemos decidido dejar como *index.php* directamente el *buscarruta.php* ya que, de esta manera, el usuario final entra directamente en el verdadero objetivo de la web, que es el de elegir una ruta y sus preferencias y posteriormente recibirla. Véase figura 3.41.

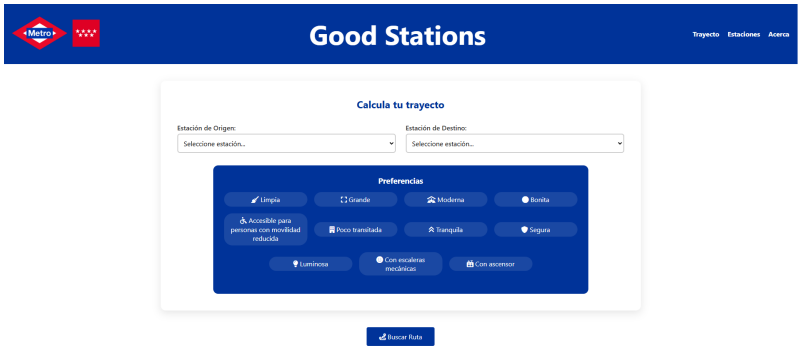

Figura 3.41: Página Inicial de nuestro proyecto

Cómo se puede apreciar en la figura 3.41 existe un considerable cambio en la parte estética de la web, haciéndola de esta manera más atractiva al usuario final. Como punto de partida hemos modificado el tipo de fuente de todo el proyecto con una fuente más moderna y algo más atractiva a nivel visual. El icono de la comunidad de Madrid de la cabecera lo hemos movido a la zona izquierda dónde se encuentra el icono del Metro de Madrid y hemos liberado la zona derecha de la cabecera dejando únicamente el menú de nuestra web. Con respecto al boceto inicial, hemos modificado toda la parte estética de *Calcula tu trayecto*, hemos añadido botones con distintos iconos que representan las distintas características seleccionables para calcular el trayecto, que pulsando sobre ellos se quedan alumbrados de forma permanente cómo si el botón se encontrara pulsado en todo momento. Cómo decimos, una versión visualmente más bonita y atractiva, y funcionalmente más intuitiva para el usuario final.

Seleccionando la estación de origen y destino y la preferencia que quiera el usuario y pulsando en el botón `Buscar Ruta`, llegamos a la página *resultadosruta.php* dónde se procesa la ruta creando un nuevo objeto y llamando a otro fichero *FormularioResultadosRuta.php*, en este fichero se encuentra una de las partes más complejas del proyecto y que más quebraderos de cabeza nos ha dado, CONECTAR el backend con el frontend, todo ello lo comentaremos en la siguiente subsección más detalladamente. Siguiendo con la parte estética, en esta página se mostrará una serie de rutas disponibles en función de la característica seleccionada en el formulario anterior, pueden ser de 1 hasta 3 rutas disponibles. Las distintas opciones de ruta son

mostradas en distintos cuadros en el centro de la pantalla en el que se muestra información del tiempo, las líneas, las preferencias seleccionadas y los transbordos. Para ver la ruta completa, hemos incorporado un botón `Ver detalles` que despliega el cuadro indicado mostrándote la ruta desde el origen al destino pasando por las distintas estaciones e indicando los minutos de distancia que hay entre cada estación. Véase figura 3.42.

Cómo proyecto futuro se quiere incorporar un mapa que pinte la ruta al estilo Google Maps, ya que de esta forma daría un toque aún más intuitivo y elegante a la web y daría un salto de calidad considerable a nuestro proyecto.

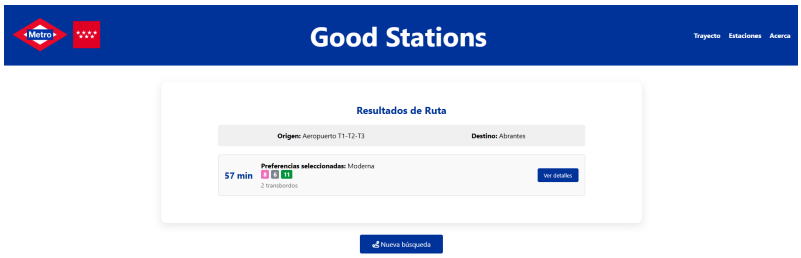

Figura 3.42: Página dónde se muestra el resultado de la ruta

La otra parte clave de nuestra aplicación web, reside en la sección de `Estaciones`, que cómo en el boceto inicial muestra las distintas líneas que existen y pinchando en cada una, muestra las estaciones que tiene cada línea y su correspondiente descripción. La versión final ha sido mejorada estéticamente de forma considerable ya que se han incorporado diferentes recursos y técnicas CSS para que la página sea más agradable y elegante para el usuario. Véase figura 3.43.

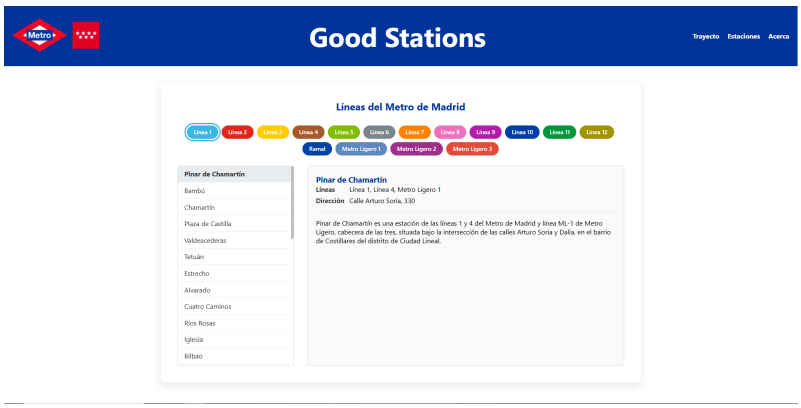

Figura 3.43: Página dónde se muestra el listado de estaciones

Al igual que *index.php* como *resultadosruta.php* la sección de `Estaciones` se encuentra definido en el fichero *lineasmetro.php* y tiene aplicado la técnica de encapsulación de objetos ya que este fichero tiene definido *new Lineas()* que crea el objeto y procesa la información en *Lineas.php*.

Para finalizar, tenemos el menú `Acerca` que cómo dijimos en secciones anteriores muestra una descripción de nuestro proyecto, todo esto definido sobre el fichero *acerca.php*. Véase figura 3.44.

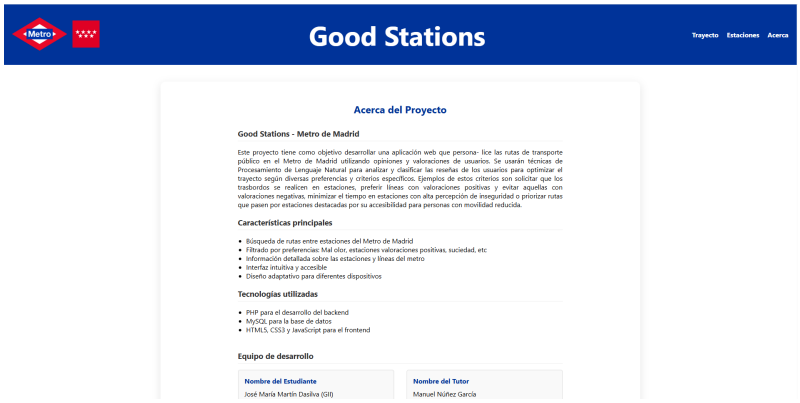

Figura 3.44: Página de Acerca del proyecto

3.9.3. Incorporación y conexión de la API en la Web (Front-end)

Para implementar la lógica de cálculo de rutas entre estaciones del Metro de Madrid, se optó por realizar una conexión con una API del backend. Toda la lógica relacionada con la búsqueda y filtrado de rutas está centralizada en la API desarrollada en Python (servidor), mientras que la aplicación PHP funciona únicamente como cliente, encargándose de gestionar la interacción con el usuario y mostrar los resultados.

Desde PHP, el punto de entrada para obtener las rutas es la función *llamaApiRutas()*. Esta función se encarga de construir la consulta a la API en función de los datos que el usuario ha introducido en el formulario: estación de origen, estación de destino y distintas características. Para construir la URL final con estos parámetros, se utiliza la función *httpbuildquery()* , a la que posteriormente se le añaden manualmente los filtros que entran por parámetro en la función.

Una vez formada la URL de la petición, se lanza una solicitud HTTP utilizando cURL, que es una librería integrada en PHP que permite hacer peticiones web. Se configura para que devuelva directamente la respuesta como una cadena de texto y define un pequeño timeout para evitar bloqueos prolongados si la API no responde. Si todo va bien y la respuesta es válida, se exporta el JSON devuelto por la API a un array de PHP. En caso contrario (por ejemplo, si el servidor responde con un error o no se puede establecer la conexión), se registra el error en el log del servidor y se devuelve null.

Seguidamente y una vez obtenida la respuesta, se pasa por una función llamada *transformaRutas()*, que adapta el contenido a un formato orientado al frontend. Esta parte es clave, ya que no solo traduce los datos tal cual, sino que también añade información útil para mostrar en la web. Por ejemplo, se calcula automáticamente

el número de transbordos, la lista de líneas del trayecto, y se facilita el tiempo aproximado que se tarda entre cada parada del trayecto.

Además, se etiqueta si la ruta cumple con los criterios de accesibilidad (por ejemplo, si todas las estaciones tienen ascensor, en caso de que el filtro correspondiente haya sido activado), y se incluye una descripción generada por la propia API. Todo este conjunto se devuelve como un array que será posteriormente utilizado por la parte visual de la aplicación.

En cuanto a mostrar este contenido en la web, esto está completamente integrado dentro del método *generaCamposFormulario()*. Esta función se encarga de recorrer las rutas recibidas y generar dinámicamente el código HTML que se mostrará al usuario. Para cada ruta, se crea un cuadro que muestra la información antes mencionada en la sección anterior como el tiempo total del trayecto, el número de transbordos, las líneas de metro que se utilizan, y un icono de accesibilidad en caso de que la ruta sea totalmente apta para personas con movilidad reducida.

Cómo hemos comentando anteriormente, cada cuadro incluye además el botón `Ver detalles`, donde pinchando se muestra el recorrido completo de la ruta paso a paso. En esta parte, se incluye también el tiempo estimado entre una estación y otra, lo cual proporciona un nivel de detalle muy atractivo y mejora la experiencia del usuario. A nivel técnico, este despliegue se gestiona con un pequeño script JavaScript que añade o quita una clase visible al bloque de detalles correspondiente, haciendo que se muestre u oculte sin necesidad de recargar la página, este script está integrado en *resultadosRuta.js*.

Esta arquitectura no solo permite separar de forma clara la lógica de cálculo (API) de la lógica de presentación (PHP), sino que también mejora la escalabilidad del proyecto. En el futuro, si se quisiera cambiar el sistema de cálculo de rutas o integrarlo con otros sistemas externos, bastaría con modificar la API, sin tocar la parte del frontend.

En resumen, esta combinación de PHP como capa de presentación y API como lógica nos ha permitido construir una solución modular, clara y fácilmente extensible, adaptada a las necesidades del usuario y respetando el diseño y la experiencia de uso propias de una aplicación real de transporte público.

3.9.4. Valoración final de la Web con usuarios finales

El propósito de realizar una evaluación final de nuestra aplicación es valorar la interactividad, accesibilidad y usabilidad de nuestra aplicación web en su versión definitiva para saber y conocer la opinión de los usuarios acerca de nuestro trabajo.

Se realizará una evaluación con usuarios utilizando el producto final de nuestra web en un pc con la aplicación accesible.

3.9.4.1. Preguntas de evaluación

Las preguntas serán similares a las realizadas en las entrevistas realizadas en fase de investigación pero con un objetivo y una valoración distinta:

- ¿Los usuarios pueden navegar de forma fácil por la plataforma?
- ¿Los usuarios saben qué elementos son accionables y cuáles no?
- ¿Los usuarios tienen suficiente control y libertad?
- ¿Se pueden realizar las acciones importantes con pocas pulsaciones?
- ¿El usuario es capaz de interactuar con la aplicación de manera correcta?
- ¿Los flujos de la aplicación son fáciles de entender?

3.9.4.2. Listas de tareas a realizar

El entrevistador irá diciendo una serie de tareas para que el usuario, sin ayuda, las realice y nos vaya indicando su opinión:

- Planificar una ruta
  - Seleccionar una estación de origen
  - Seleccionar una estación de destino
  - Seleccionar una preferencia o característica
- Revisa los resultados al calcular ruta
  - Selecciona una de las rutas
  - Despliega la ruta
  - Calcula otra ruta
- Menú de navegación
  - Usar el menú de navegación para ver las diferentes páginas que contiene la web
- Estaciones
  - Ir usando el menú a la página de Estaciones
  - Seleccionar una estación cualquiera y mirar su información
  - Mirar una estación de una línea distinta para ver la navegación entre líneas
- Acerca
  - Ir a esta página para leer información sobre el proyecto contenida en la web

3.9.4.3. Participantes de la entrevista y valoraciones

La selección de los participantes ha sido la misma que en la evaluación del boceto, ya que buscábamos que el mismo usuario nos diera su punto de vista y describiera los cambios que había visto con respecto al boceto.

Ana Gallego

La entrevista realizada a Ana ha vuelto a ser bastante útil para nosotros. En pocas palabras a la usuaria le ha *encantado* la web, dicho con esas palabras. Según la usuaria, se trata de una web bonita, muy intuitiva y con los objetivos claramente marcados.

Ana comenta que ha podido navegar de forma fácil por toda la web, no ha encontrado fallos mientras utilizaba la aplicación y sabe perfectamente qué elementos existen, cuáles son accionables y cuáles no.

La entrevistada nos comenta que la aplicación tiene bastante más ítems y características que el boceto y le ha parecido que la aplicación está bastante cambiada y mejorada al respecto.

Desde el perfil de enfermera nos ha vuelto recalcar la característica que muestra si las estaciones tienen accesibilidad para personas con discapacidad. Considera que esta función está muy bien integrada y la destaca como algo muy positivo para personas de ese tipo. Igualmente nos indica que las características de Acensor o escaleras mecánicas son bastantes importantes y son muy útiles para saber que rutas realizar con este tipo de preferencia.

En definitiva, la experiencia de la usuaria ha sido muy positiva y no pone ningún pero a nuestra aplicación web final.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

Alejandro Martín

La entrevista al usuario Alejandro Martín en cuánto a valoraciones y resultados es similar a la opinión de Ana. Su opinión a nivel global ha sido que la aplicación web cumple con las expectativas del usuario.

El usuario ha hecho varias pruebas calculando varias rutas y ha recalcado que la funcionalidad de la web y la búsqueda de rutas es muy intuitiva y fácil de realizar, para todos los públicos.

En cuanto al entorno gráfico y la estética de la web el usuario recalca en varias ocasiones que la web es muy bonita y elegante a nivel visual, con los colores perfectamente definidos para las líneas de metro como para el entorno en general.

En las preguntas finales tras la realización de las tareas, el usuario recalca que ha podido navegar de forma muy sencilla por la plataforma, que ha tenido control y libertad suficiente en la web, ha podido realizar las acciones importantes con pocas

pulsaciones, ha sido capaz de interactuar con la aplicación de manera sencilla y los flujos son muy fáciles de entender.

En conclusión, la experiencia del usuario ha sido muy positiva y deja un mensaje claro, la aplicación va ayudar a la población.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

Fan Yen

La entrevista al usuario Fan Ye ha sido bastante útil y productiva para nuestra valoración final de la web ya que desde su perfil de antiguo alumno nos ha otorgado una visión más técnica de la web.

El usuario ha realizado varias pruebas calculando varias rutas, una más corta y otra más larga, para comprobar los distintos casos y ver cómo actúa la web a nivel funcional y estético. El usuario ha recalcado que la funcionalidad de la web y la búsqueda de rutas es muy intuitiva y estética muy bonita.

En cuanto al entorno gráfico y la estética de la web el usuario indica en varias ocasiones que la web tiene una estética minimalista y fácil de entender.

El usuario indica que ha podido navegar de forma muy fácil, libre y ha podido realizar las acciones importantes con pocas pulsaciones y ha sido capaz de interactuar con la aplicación de manera sencilla y los flujos son muy fáciles de entender.

El usuario indica que por poner un pequeño pero, cuándo la ruta es demasiado larga no se aprovecha del todo el cuadro y hay que hacer scroll hacia abajo que resultado algo incómodo para el usuario.

En definitiva, la experiencia del usuario ha sido muy positiva y nos ha indicado pequeñas críticas constructivas para futuros cambios de la web.

Adjuntamos enlace con la entrevista: Acceso a la entrevista

Capítulo 4

Contribuciones personales

José María Martín Dasilva

A lo largo del desarrollo de este trabajo, he participado de forma proactiva en varias de las etapas de este proyecto, empezando desde la obtención y tratamiento de datos, hasta del diseño, desarrollo y evaluación de la aplicación web. Mis aportaciones al proyecto han formado parte tanto de aspectos más técnicos como de diseño y pre-evaluación y evaluación, lo cual me ha permitido implicarme de forma global en todo el trabajo. La primera tarea en la que trabajamos en grupo fue en la obtención de datos mediante Web Scraping, centrada en recoger reseñas reales de los usuarios sobre las estaciones del Metro de Madrid. Junto a mis compañeros, nos organizamos para dividir las 275 estaciones de metro de forma equitativa entre los cuatro integrantes del equipo, de este modo cada uno se encargaba de un número de estaciones específicas. Para extraer los datos de cada estación, utilicé la extensión Instant Data Scraper, que instalé como extensión en mi navegador Google Chrome, lo cual me permitió automatizar "en cierta manera" la recolección de los datos desde los distintos enlaces de Google Maps de las estaciones. Finalmente, almacené junto con mis compañeros el conjunto de datos en nuestro repositorio común de Google Drive, y así facilitar el acceso del equipo a la información.

Una vez recopilados los datos anterior, empecé con en el análisis de herramientas de diseño de interfaces y prototipado web. Con el objetivo de seleccionar la plataforma ideal para el desarrollo de nuestro prototipo, realicé una comparativa entre varias herramientas destacadas del sector: Sketch, Adobe XD, InVision, Framer, Axure RP, Canva, Proto.io, Balsamiq, Gravit Designer y Figma. Mediante artículos web y vídeos analicé varios aspectos como la curva de aprendizaje, las funcionalidades de prototipado, la colaboración en equipo y la compatibilidad multiplataforma de las diferentes plataformas. Tras esta evaluación, finalmente me decanté por Figma, una herramienta más completa, más versátil y accesible que permite diseñar de forma colaborativa desde el navegador. En Figma llevé a cabo la creación del boceto web de nuestra aplicación, incluyendo los distintos flujos de navegación y simulando la experiencia de usuario que queríamos ofrecer.

Posteriormente, empecé con la fase de evaluación del boceto mediante entrevistas a usuarios reales. En mi caso, me encargué tanto de realizar las entrevistas como de grabarlas, ejerciendo simultáneamente de entrevistador y de cámara. Las entrevistas las realicé sobre dos usuarios, Ana Gallego y Alejandro Martín, con el objetivo de recoger sus impresiones al interactuar con el boceto. Tras cada entrevista, procedí a realizar la transcripción completa de las respuestas. Las transcripciones las compartí con el equipo a través de nuestro repositorio en Google Drive. A partir de estas entrevistas, saqué una serie de conclusiones que nos permitieron detectar aciertos y posibles mejoras en nuestro diseño, y así procurar mejorar de una forma más precisa nuestra aplicación web.

Posteriormente, empecé con el diseño y la de la base de datos del proyecto. Entre ello, me encargué de recopilar información detallada línea por línea de cada estación desde la web oficial del Metro de Madrid. Con esta información, fui creando manualmente la tabla principal de estaciones (un total de 275), definiendo campos como el ID, el nombre, una breve descripción y la dirección de cada una. De igual forma, colaboré en la creación de las demás tablas necesarias para estructurar correctamente nuestra base de datos: lineas, estaciones_lineas, características, conexiones y estaciones_caracteristicas, con sus datos correspondientes a cada línea.

Por último, me encargué de la creación del proyecto web final. Procedí junto a mi compañero Daniel a estructurar todo el sistema de archivos y carpetas del proyecto, estableciendo una estructura clara que separara la lógica de la web, los recursos visuales, los estilos, los scripts y las vistas. También desarrollé el código de la aplicación utilizando PHP, y diseñé la parte visual mediante hojas de estilo CSS y generé varios scripts de JavaScript que permite validar ciertos datos a la hora de elegir estaciones proyectando información dinámica sobre la web. Programé las distintas funcionalidades de la aplicación: desde el diseño y la visualización de los formularios para elegir estaciones y características, cómo la web dónde se muestra el resultado de la ruta, cómo página que exporta información de cada línea y sus estaciones, cómo la página de acerca.

Una de las partes que más quebraderos de cabeza me ha dado ha sido la integración de la información del backend en la web. Para ello consulté y busqué información para realizarlo y tras varios días de estudio conseguí realizar las funciones descritas en la sección 3.9.3 y así poder integrar la lógica exportada por la API del backend en nuestra web. El diseño CSS y la lógica PHP y HTML de esta parte también ha sido íntegramente realizado por mí.

En cuánto a las entrevistas finales realizadas a los usuarios, me encargué de diseñar las preguntas y las tareas a realizar, y me ocupé de realizar las 3 entrevistas Ana, Alejandro y Fan Ye. En este caso, me encargé de grabar, entrevistar, transcribir la entrevista y subirla a nuestro repositorio de google drive. Las entrevistas han sido incluídas en la memoria en formato de enlace.

Para finalizar, en cuándo al desarrollo de la memoria del trabajo en mi caso me he ocupado de prácticamente todos los puntos desde la sección 3.7 hasta la subsección 3.9.4.3, excepto la sección 3.8 que ha sido desarrollada por mi compañero Daniel. Por otra parte he contribuido en ayudar en otras secciones de la memoria cómo el

resumen, palabras clave, introducción y estado del arte, al igual que incluí varios puntos de la bibliografía.

En resumen, mi participación en este trabajo ha sido importante, aportando desde la fase inicial de recolección de datos hasta el desarrollo completo de la aplicación web. He contribuido tanto en el análisis y diseño como en la implementación técnica y la evaluación final, lo que me ha permitido aprender y aplicar conocimientos muy diversos en este trabajo.

Ibon Malles Altolaguirre

Inicié mi participación en el proyecto escogiendo de qué plataformas íbamos a extraer las reseñas de usuarios (aunque se estudiaron numerosas opciones solo se escogió Google Maps). Seguidamente, junto con mi compañero Francisco y, escogimos qué herramienta se iba a utilizar para la extracción de las reseñas.

Al igual que el resto del equipo, llevé a cabo el scraping de las reseñas correspondientes a una cuarta parte de las estaciones de metro de Madrid. Después, desarrollé un programa para analizar todas las reseñas y detectar errores introducidos por `Instant Data Scraper`. El trabajo de corrección de estos errores lo repartimos entre todos los integrantes del grupo.

A partir de este momento nos dividimos en dos grupos: Francisco y yo en uno y Daniel y José María en otro.

Una vez tuvimos todas las reseñas extraídas y procesadas, Francisco y yo nos encargamos de etiquetar un subconjunto de ellas con el objetivo de construir una base de datos que nos permitiera entrenar un clasificador. Dado que contábamos con un volumen limitado de datos, investigué técnicas de aumento de datos textuales. Nos dividimos las metodologías encontradas: yo me centré en aplicar técnicas de "retrotraducción" y del uso de modelos de última generación vía API.

Aunque la implementación del aumento de datos con modelos preentrenados la realizamos conjuntamente Francisco y yo, asumí la parte de investigación sobre qué modelos eran más apropiados, sus ventajas y los posibles retos que podían surgir. Además, definí el enfoque para comparar las distintas técnicas, estableciendo métricas que midieran la similitud léxica y semántica entre los textos generados y los originales. Para finalizar con la sección del aumento de datos me encargué de analizar y comparar todos los métodos entre sí.

Me encargué íntegramente del diseño y entrenamiento del clasificador. Tras investigar las distintas metodologías disponibles, concluí que la opción más adecuada era emplear modelos preentrenados y ajustarlos a nuestro caso de uso. Dado que disponíamos de dos conjuntos de datos (las reseñas originales y las generadas mediante técnicas de aumento), decidí realizar un estudio para evaluar la utilidad de los datos sintéticos en el proceso de entrenamiento.

Con este objetivo, entrené tres clasificadores siguiendo estrategias distintas en cada caso. El primero, que obtuvo los mejores resultados, fue entrenado en una única

fase combinando datos originales y sintéticos. El segundo se entrenó en dos etapas diferenciadas utilizando ambos conjuntos de datos de manera secuencial. Por último, el tercero se entrenó exclusivamente con datos originales en una única fase.

Con los tres modelos disponibles, me encargué de realizar un estudio de cada uno de ellos, extrayendo métricas objetivas, y después los comparé y escogí el que mejores resultados dio. Consecuentemente, fui capaz de concluir que el uso de datos sintéticos aumenta ligeramente el rendimiento de los modelos.

Con los tres clasificadores ya entrenados, llevé a cabo un análisis detallado de su rendimiento. Para ello, extraje métricas objetivas como la precisión, el recall y la F1-score, lo que me permitió realizar una comparación fundamentada entre los distintos enfoques. Tras este estudio, seleccioné el modelo que ofrecía un mejor equilibrio entre las métricas evaluadas y, por tanto, un rendimiento más sólido y consistente. Como resultado, pude concluir que la incorporación de datos sintéticos, aunque no produce una mejora drástica, sí aporta una ligera mejora en el rendimiento general del modelo.

La primera mitad de la extracción de las características de las estaciones de metro fue implementada en su mayoría por Francisco, únicamente contribuí en el estudio de la librería `spaCy` y con algunos detalles de la extracción (casos extremos). La segunda mitad de esta sección la realizamos en su mayoría de forma conjunta entre Francisco y yo. Ambos nos encargamos de diseñar el proceso para la obtención de las características de todas las estaciones, de combinar aquellos sustantivos que fuesen sinónimos, para lo cual tuvimos que diseñar una función para la detección de palabras con similitud semántica elevada usando grandes modelos de lenguaje y la similitud entre las representaciones vectoriales y de combinar y filtrar los adjetivos. Por otra parte, el filtrado de sustantivos lo realicé por mi cuenta. Finalmente, aplicamos el "pipeline" descrito sobre todas las estaciones para obtener el listado de las características finales.

El estudio inicial sobre la librería más adecuada para representar la red de Metro de Madrid fue llevado a cabo por Francisco, quien identificó la librería `Networkx` como una opción viable. A partir de su propuesta, me encargué de analizarla en mayor profundidad, evaluando sus capacidades y limitaciones en nuestro contexto, así como de investigar otras alternativas que pudieran adaptarse mejor a nuestras necesidades. En paralelo, exploré enfoques distintos para la representación de grafos, como estructuras basadas en diccionarios o modelos inspirados en máquinas de estados.

Dado que esta parte constituía el núcleo funcional de la aplicación, Francisco y yo la desarrollamos conjuntamente en su totalidad. Determinamos el tipo de grafo más adecuado para nuestro caso, definimos la estructura de los nodos y las aristas, y diseñamos las distintas estrategias de cálculo de rutas. Estas incluían: un enfoque en el que se eliminaban directamente los nodos que no cumplían con los filtros establecidos; otro en el que únicamente se descartaban estaciones de transbordo; y una tercera estrategia en la que las aristas correspondientes a transbordos eran evaluadas dinámicamente según los parámetros introducidos por el usuario.

Por último, nos ocupamos de adaptar los resultados generados por el algoritmo

para que el equipo de *frontend* pudiera integrarlos sin dificultad en la interfaz. Esto implicó la generación de todas las rutas posibles, el listado ordenado de estaciones, los tiempos estimados de recorrido y la inclusión de descripciones detalladas que facilitasen la comprensión del trayecto por parte del usuario.

Como parte final del desarrollo de la aplicación, Francisco y yo diseñamos y desarrollamos una API destinada a centralizar la lógica de optimización de rutas y a facilitar la comunicación entre el *frontend* y el *backend*. Esta API no solo permitió encapsular el funcionamiento interno del sistema de cálculo de rutas, sino que también aportó modularidad al proyecto, facilitando futuras ampliaciones y el mantenimiento del código. Nos encargamos de definir una estructura clara y coherente para los endpoints y de estandarizar los formatos de entrada y salida. Gracias a ello, el equipo de *frontend* pudo integrar esta funcionalidad en la interfaz sin dificultades y de forma eficiente.

Además de desarrollar las tareas que me correspondían directamente, asumí la responsabilidad de diseñar la base de datos general del proyecto, asegurándome de que su estructura respondiera adecuadamente a los requisitos del sistema. También colaboré de forma activa con el equipo formado por José María y Daniel, apoyándoles siempre que fue necesario. En concreto, Francisco y yo identificamos errores en la introducción de datos, proporcionamos comentarios constantes sobre aspectos de la interfaz web, especialmente en lo relativo a la estética y usabilidad, y recomendamos herramientas auxiliares que facilitaron su trabajo e incrementaron la cohesión del sistema.

Finalmente, la redacción de la memoria del proyecto correspondiente a las secciones que habíamos implementado Francisco y yo la dividimos equitativamente. Cada uno se encargó de redactar la mitad del contenido, revisando cuidadosamente el trabajo del otro para garantizar la coherencia y calidad del documento final. Además, asumí la responsabilidad de redactar el apartado de conclusiones y trabajo futuro, sintetizando los aprendizajes obtenidos a lo largo del desarrollo del proyecto y proponiendo posibles líneas de mejora y ampliación para trabajos posteriores.

Francisco Prieto Gallego

Mi trabajo a lo largo del desarrollo del proyecto ha sido bastante activo. Todo empezó con la búsqueda de alguna herramienta para poder extraer las reseñas que los usuarios escribían a través de distintas aplicaciones y redes sociales. Tras mirar y comparar varias opciones junto a mis compañeros la que acabamos utilizando fue una extensión de Google Chrome que nos recomendaron los tutores.

Al igual que el resto de mis compañeros realicé el *scrapping* de las sesenta y nueve estaciones que me tocaron para poder llevar a cabo la primera parte del proyecto, la recaudación de datos a través de las reseñas realizadas por los usuarios en Google Maps. A continuación, tras un *script* realizado por Ibon para detectar errores en el conjunto de reseñas obtenidas procedí a la corrección de mis estaciones.

Tras realizar el *scrapping* y la limpieza de las reseñas, nos dividimos en dos

grupos, mis compañeros José María y Daniel se encargarían de la parte del *frontend* y paralelamente mi compañero Ibon y yo nos encargaríamos del *backend*.

Una vez realizada la división de equipos Ibon y yo nos dimos cuenta de que no todas las reseñas obtenidas eran útiles. Debido a esto decidimos crear un clasificador para determinar las reseñas útiles. Para ello, primero nos dedicamos a clasificar a mano un porcentaje de ellas para divirlas y etiquetarlas entre válidas y no válidas. A continuación, notamos como el conjunto de datos disponibles era muy reducido para generar cualquier tipo de clasificador con una precisión aceptable, por lo que tras un estudio previo de mi compañero Ibon decidimos realizar un aumento de datos mediante distintas técnicas. Para esta parte nos dividimos las técnicas a implementar cada uno. Yo me encargué de implementar las técnicas de reemplazo por sinónimos, inserción aleatoria, intercambio aleatorio, eliminación aleatoria, albumentación y una combinación de todas ellas. Además, tras otro estudio previo de mi compañero para complementar estas técnicas decidimos utilizar modelos preentrenados para poder realizar un parafraseo y así seguir aumentando el conjunto de datos que habíamos obtenido. Cabe destacar que tanto la investigación sobre todas estas técnicas como la elección sobre que modelos usar fue todo realizado por mi compañero pero la implementación de ellas fue llevado a cabo por los dos en conjunto.

Tras la realización del aumento de datos Ibon se encargó de la realización del clasificador y yo mientras tanto de ir implementando la extracción de características. Para ello realicé un estudio de las distintas técnicas para poder llevar a cabo esta parte. Las técnicas que estudié fueron el uso de N-Gramas y librerías basadas en redes neuronales. Tras investigar a fondo cada una de ellas realicé su implementación sobre un conjunto de reseñas de prueba para poder compararlas y elegir la que mejor resultados obtuviese. Una vez Ibon terminó el clasificador yo tenía ya implementado todas las técnicas y realizado el estudio comparativo para quedarnos con la mejor. Sin embargo, mi código tenía algún pequeño problema y entre ambos los corregimos. A continuación, y nuevamente entre ambos, utilizando las reseñas filtradas por el clasificador obtuvimos un diccionario con todas las características de las distintas estaciones de Metro de Madrid.

Después de la obtención de las características realizamos un estudio sobre los resultados obtenidos. Nos dimos cuenta de como había características distintas con el mismo significado y como había características distintas con significados opuestos para una misma estación. Para evitar esta clase de incongruencias decidimos primero agrupar aquellas características sinónimas en una misma estación mediante embeddings lingüísticos. Esta agrupación se llevo a acabo en dos partes, la primera en unir los sustantivos que tuviesen el mismo significado, realizado individualmente por mi compañero Ibon y la segunda en agrupar los adjetivos sinónimos relacionados a cada sustantivo, implementado por los dos. Una vez estaban filtradas en todas las estaciones aquellas características con mismo significado me encargué de eliminar aquellos que tuviesen un significado opuesto. Para ello intenté en una primera instancia usar las distintas librerías de Python encargadas para poder eliminar antónimos. Sin embargo, el poder realizar esta tarea en español conllevaba a invertir una cantidad de tiempo de procesamiento inviable, por lo que tuve que ir estación

a estación comprobando cada una de las características y eliminándolas a mano.

Una vez filtradas todas las características me encargué de investigar distintas librerías y métodos para poder representar el grafo del Metro de Madrid y así poder calcular las rutas. La que más me gustó fue la de `Networkx`. Ibon se encargó de buscar otras alternativas pero ambos acordamos finalmente en utilizar esa. Tras obtener la librería con la que íbamos a trabajar llevamos a cabo en conjunto todas las decisiones acerca del tipo de grafo que íbamos a usar, la representación del mismo (definición de nodos y aristas) y las distintas estrategias a usar para poder calcular las rutas. Una vez decidido todo, implementamos la construcción del grafo, tres estrategias distintas para poder calcular la mejor ruta posible desde tres enfoques distintos y por último adaptar nuestros resultados tras aplicar los algoritmos para que la otra mitad del grupo pudiese incorporarlos sin muchas complicaciones en la página web. Esto se realizó devolviendo un JSON para que pudiesen adaptarlo en su parte.

Por último, Ibon y yo nos encargamos de implementar una API para poder centralizar la lógica de optimización de rutas y facilitar la comunicación entre el *frontend* y el *backend*. Esto lo realizamos a través de una librería de Python que encontró Ibon. La creación de esta API ayudó tanto al equipo de *backend* para poder encapsular el funcionamiento de la creación de rutas como al equipo de *frontend* la cual le permitió integrar esta funcionalidad en su pagina web sin mayores complicaciones y de forma eficiente.

Aparte de trabajar en los distintos aspectos necesarios para la implementación del *backend*, di ayuda y feedback acerca de cada uno de los pasos en la realización del *frontend* siempre que fuera necesario.

Además, como cada uno de los integrantes del trabajo tuve una contribución activa para la redacción de la memoria, escribiendo a medias junto con mi compañero Ibon de una forma equitativa la parte del proyecto que hemos realizado. Aparte, al igual que Ibon, revisé con mucho detalle siempre lo que había escrito él para que estuviese descrito de la mejor manera posible. También me encargué de buscar la información y redactar toda la parte del estado del arte y de la introducción. Por último, me encargué de revisar y corregir errores menores en la redacción de la memoria como formatos y comillas.

Daniel Pérez García

Mis aportaciones a lo largo de este proyecto han consistido de varias tareas que han ido variando a medida que ha avanzado el desarrollo de éste. Estas tareas se han centrado en la obtención y recopilación de datos para su posterior tratamiento y en el desarrollo de la página web pensada para la interacción con usuarios.

En la primera fase del desarrollo del proyecto se nos encomendó la tarea de obtener las distintas reseñas de todas las estaciones del Metro de Madrid. Para ello, utilizamos técnicas de Web Scraping para poder guardarlas en ficheros que posteriormente serían procesados. La herramienta utilizada fue Instant Data Scraper, un programa que se instala a modo de extensión en navegadores como Google Chrome.

Este programa automatiza el proceso de seleccionar las reseñas para poder descargarlas, lo cual nos ha agilizado en gran medida este proceso que incluye una gran cantidad de datos. Al estar todo el grupo realizando esta tarea, optamos por dividirnos equitativamente las estaciones de las cuales íbamos a extraer las reseñas. Una vez extraídas las reseñas, procedimos a adaptar el formato de las reseñas para poder procesarlas, eliminando elementos como algunos caracteres especiales o saltos de línea dentro de una misma reseña. Por cada estación generamos un archivo .csv, los cuales almacenamos en una carpeta de Google Drive para ponerlos en común.

Una vez que terminamos la extracción de las reseñas, nos dividimos en 2 grupos para realizar el resto de partes que conforman el proyecto. En mi caso, estuve trabajando junto con José María en el diseño de la página web. Para empezar con esta parte, decidimos hacer un prototipo de interfaz de usuario que incluyera tanto los elementos que iban a conformar la página web como un primer diseño del aspecto que iba a tener la web. Mi compañero estuvo mirando distintas herramientas, pero finalmente nos decantamos por utilizar Figma, ya que es una herramienta muy completa que te permite realizar tanto bocetos como prototipos interactivos de una manera intuitiva y fácil de usar. Además, Figma era una herramienta que había estado usando en la asignatura de Desarrollo de Sistemas Interactivos y estaba más familiarizado con ella. En Figma definimos todas las páginas que iba a tener la web y qué elementos iba a tener. Además, hemos definido los flujos de la web para poder hacerse a la idea de su funcionamiento.

A continuación, procedimos con la programación de la página web. Primero que todo, teníamos que decidir qué lenguaje o lenguajes de programación íbamos a utilizar para construirla. Al principio decidimos que la íbamos a implementar utilizando React, una biblioteca de Javascript. Estuvimos investigando sobre ese lenguaje mediante sitios web que contenían explicaciones básicas o videotutoriales en Youtube. Sin embargo, decidimos rápidamente desechar esa idea, ya que pensábamos que nos iba a suponer un esfuerzo mayor el hecho de tener que investigar y aprender a cómo realizar correctamente una página web entera utilizando esa tecnología. Por lo tanto, decidimos utilizar HTML y PHP para programar la web, ya que son lenguajes que hemos utilizado tanto en la asignatura de Aplicaciones Web como en la de Ampliación de Bases de Datos y estábamos mucho más familiarizados con ellos.

Cuando ya establecimos la tecnología que íbamos a usar, procedí a estructurar y organizar los archivos que iban a formar parte de la página web. Para empezar, decidí separar la lógica de la web de la vista. En el directorio raíz de la carpeta de la Web, se incluyen todos los archivos que incluyen la vista de la web. Para los archivos de lógica, creé varias carpetas en función de la funcionalidad de cada archivo. Los directorios que componen la web son: *css*, que contiene la hoja de estilos de la aplicación; *images*, en el que se encuentran las imágenes que se incluyen en la aplicación; *js*, que tiene los diferentes scripts que utilizamos en la web; y por último, *includes*, que está conformado por otros 3 directorios: *db*, donde guardamos los archivos que conforman la base de datos; *vistas*, que incluye dos archivos que sirven como plantilla para la estructura visual de la web, y *src*, que contiene los archivos que conforman la lógica de la aplicación.

Para la base de datos, mi compañero José María y yo nos pusimos a rellenar de

datos las tablas que Ibon nos especificó que eran necesarias para que él y Francisco pudieran procesar los datos que necesitaban para realizar el backend. Establecimos identificadores numéricos para cada estación, línea y característica y fuimos completando la base de datos apoyándonos en fuentes como la web oficial del Metro de Madrid o la Wikipedia. Para completarla, decidimos usar la herramienta phpmyadmin, que viene incluida con la herramienta XAMPP, ya que ésta nos ha permitido darle el formato necesario a la base de datos. Este formato se compone de la composición de las columnas de las tablas con sus respectivos tipos, la definición de las claves primarias y las relaciones que iban a tener las tablas entre sí, ya que la hemos realizado en SQL, que es un lenguaje para bases de datos relacionales, para completar así la estructura de la base de datos. Además, gracias a phpmyadmin, se nos facilitaba el hecho de introducirle los datos, ya que al usar datos numéricos que se autoincrementaban, phpmyadmin ya nos cubría esa funcionalidad y aceleraba ese proceso.

Una vez teníamos una versión provisional de la base de datos y la página web, José María y yo procedimos a realizar evaluaciones con usuarios basándonos en el boceto que habíamos realizado en Figma para que nos diesen feedback de cara a la versión final del proyecto. Estas evaluaciones las realizamos aplicando técnicas que habíamos aprendido en la asignatura de Desarrollo de Sistemas Interactivos. Por mi parte, yo realicé la entrevista a Fan Ye, mientras que mi compañero se encargó de las otras dos. Después de las entrevistas, procedimos a su posterior análisis para poder sacar conclusiones sobre cómo tenía que variar el diseño de la web.

Más tarde, después de recibir el feedback, procedimos a adaptar la web a los cambios que nos habían sugerido nuestros entrevistados en cuanto a diseño se refiere. Pusimos en común los comentarios recibidos y realizamos una lluvia de ideas para introducir también nuevos elementos a la página web para hacerla todavía más accesible para futuros usuarios.

Capítulo 5

Conclusiones y Trabajo Futuro

En esta sección se recoge las principales conclusiones obtenidas a lo largo del desarrollo del proyecto, evaluando en qué medida se han alcanzado los objetivos propuestos y analizando los resultados obtenidos. Además, se presentan algunas ideas y posibles líneas de mejora que podrían explorarse en el futuro para ampliar o perfeccionar el trabajo realizado.

5.1. Conclusiones

En este Trabajo de Fin de Grado se ha desarrollado con éxito una aplicación web capaz de recomendar rutas en el Metro de Madrid, dadas las estaciones de origen, destino y una serie de filtros introducidos por el usuario. Para ello, se implementó un sistema de extracción de reseñas desde Google Maps (única plataforma encontrada con reseñas de usuarios de estaciones de metro de Madrid), que permitió construir un corpus de aproximadamente diez y siete mil opiniones de doscientas setenta y cinco estaciones de metro.

Tras obtener las reseñas, surgió la necesidad de diseñar un clasificador para determinar cuales de las opiniones almacenadas serían realmente útiles. Consecuentemente, se clasificó a mano un conjunto muy reducido del corpus para poder entrenar el modelo. Debido a la poca cantidad de datos etiquetados, se aplicaron numerosos métodos de ampliación de datos textuales, entre ellos la retrotraducción, el reemplazo por sinónimos y la albumentación. Seguidamente, se entrenaron varios clasificadores utilizando distintas técnicas y distintos datos (datos originales y/o sintéticos), se compararon los rendimientos de cada modelo diseñado y se escogió el de mejor rendimiento.

El clasificador se utilizó para filtrar aquellas reseñas de las cuales se extraerían las características de las estaciones. Aplicando distintas técnicas de procesamiento de lenguaje natural se obtuvieron para cada estación una lista de sustantivos (estación, personal, metro, escalera, servicio ...) y sus respectivas características (bonita, grande, moderna, fea, accesible ...).

Todos estos datos se introdujeron a una base de datos, y sirvieron para determinar qué filtros podría escoger el usuario al utilizar la aplicación.

Con toda esta información disponible, se diseñó el backend de la aplicación: un grafo que representa la red de metro de Madrid, donde cada nodo dispone de las características obtenidas en el anterior paso. Además, se habilitó la evaluación dinámica de las aristas del grafo dependiendo de los filtros introducidos por el usuario y se generó un algoritmo para obtener varias rutas intentando maximizar el cumplimiento de los filtros.

En paralelo, se realizó el prototipado de la página web, se evaluó con voluntarios reales y finalmente se se diseñó una aplicación simple, intuitiva y fácil de utilizar para el usuario.

No obstante, este proyecto presenta algunas limitaciones. En primer lugar, la calidad y la disponibilidad de reseñas en Google Maps varía notablemente entre estaciones, por lo que la fiabilidad de los filtros depende directamente del volumen de opiniones recogidas. En segundo lugar, el número de filtros disponibles para la elección del usuario es reducido (únicamente once). Finalmente, la aplicación no contempla variaciones dinámicas del servicio de Metro de Madrid como retrasos puntuales, incidencias en las líneas o cambios de frecuencia de paso, ya que se basa en datos estáticos de reseñas y no en información en tiempo real.

5.2. Trabajo Futuro

Con vistas al futuro, se plantean varias líneas de mejora. Sería enriquecedor ampliar el número de filtros de los que dispone el usuario, o incluso dejar la opción de texto libre. Además, para aumentar el potencial de la aplicación, sería recomendable habilitar la posibilidad de que los propios usuarios puedan introducir reseñas directamente desde la web. Esto permitiría una retroalimentación continua y el enriquecimiento progresivo de la base de datos, mejorando la precisión y actualidad de las recomendaciones. Además, si se dispusiese de más reseñas, se podría reentrenar el clasificador para poder filtrar con más precisión las opiniones introducidas.

Actualmente el grafo que representa la red de Metro de Madrid es un grafo no dirigido, lo que implica que se asume que el tiempo de desplazamiento entre dos estaciones es el mismo en ambos sentidos. Consecuentemente, una mejora sería representar la red como un grafo dirigido, lo cual implicaría ajustar los algoritmos de búsqueda.

Por último, en cuanto a la web, sería conveniente introducir un mapa de Madrid sobre el cual se plasmasen las rutas devueltas por el algoritmo. Además, se podría introducir una sección para incidencias en tiempo real sobre la red de metro.

En definitiva, este TFG ha supuesto un ejercicio completo de ingeniería informática, abarcando desde la recolección y el procesamiento de datos, el diseño de modelos de inteligencia artificial para clasificación de textos, el diseño y el uso de grafos para el cálculo de rutas, hasta el diseño y la validación de una interfaz de usuario.

Chapter 6

Introduction

"Everything seems impossible until you do it."
— Nelson Mandela

This project aims to develop a web application that personalizes public-transport routes on the Madrid Metro network, based on ratings and opinions provided by the users themselves. Natural Language Processing (NLP) techniques will be applied to analyse and automatically classify the reviews, allowing routes to be adapted to the different preferences and criteria defined by each user.

6.1. Structure of the report

The document is organised into five parts to make it easier to read:

- **Introduction**: Presents the motivation for undertaking the project, the objectives to achieve it successfully, and the working methodology the team will follow.
- **State of the Art**: Reviews technologies and applications relevant to the project, analyses previous studies, and highlights current market solutions that could compete with ours.
- **Work Description**: Details every step required to implement the final project, from data collection to development of the *frontend* and *backend*.
- **Personal Contributions**: Describes the tasks carried out by each team member.
- **Conclusions and Future Work**: Summarises the results obtained, points out any limitations, and lists goals that could not be reached or could be improved in the future.

6.2. Motivation

Public transport plays a key role in the daily lives of millions of people, especially in large cities such as Madrid. Frequent metro users have noticed that many existing route-planning apps lack personalisation options and do not always offer journeys that match individual preferences. Factors such as station cleanliness, accessibility, safety, or crowd levels are rarely considered in recommendation algorithms. This situation presents an opportunity for improvement by using reviews and ratings from metro users. The project proposes developing an application that, leveraging these user reviews, calculates personalised routes between Madrid Metro stations while integrating the filters each user considers most relevant.

6.3. Objectives

The main objective of this Bachelor's Thesis is to develop a web application that recommends routes on the Madrid Metro according to a set of filters chosen by the user. Reviews for all metro stations will be extracted from platforms such as Google Maps. A system will then process and automatically classify the opinions, extracting useful information about aspects such as cleanliness, safety, or accessibility. Based on this information, the application will offer personalised routes that match each user's selected filters. An intuitive, functional interface will also be implemented to make the tool easy for the general public to use.

6.4. Working methodology

The project will be carried out in several phases. First, the necessary data on Madrid Metro stations will be collected, including user reviews from public sources. A database will then be built to store this information in a structured manner. Next, NLP techniques will be applied to classify the reviews and extract their features. In parallel, development will be divided into two blocks: the *backend*, responsible for processing and recommendation logic, and the *frontend*, focused on the user-interface design. The team will be organised into two sub-groups that will work in coordination to ensure all system components are properly integrated. 1.

Chapter 7

Conclusions and Future Work

This section presents the main conclusions reached during the development of the project, evaluating the extent to which the proposed objectives have been achieved and analysing the results obtained. It also outlines ideas and possible avenues for improvement that could be explored in the future to expand or refine the work carried out.

7.1. Conclusions

This Bachelor's Thesis successfully developed a web application capable of recommending routes on the Madrid Metro, given the origin and destination stations and a series of user-defined filters. A review-extraction system was implemented for Google Maps—the only platform found with user reviews of Madrid Metro stations—which enabled the creation of a corpus of roughly seventeen thousand opinions covering 275 stations.

After gathering the reviews, a classifier had to be designed to determine which stored opinions would actually be useful. A very small subset of the corpus was manually labelled to train the model. Because of the limited amount of labelled data, various textual data-augmentation methods were applied, including back-translation, synonym replacement and Albumentation. Several classifiers were then trained using different techniques and datasets (original and/or synthetic), their performances compared, and the best-performing model was chosen.

The classifier was used to filter the reviews from which station features were extracted. Using different NLP techniques, each station was assigned a list of nouns (station, staff, metro, staircase, service...) and their corresponding attributes (nice, large, modern, ugly, accessible...).

All of this information was loaded into a database and served as the basis for determining which filters a user could select in the application.

With these data available, the backend was designed as a graph representing the Madrid Metro network, where each node holds the attributes obtained in the

previous step. Dynamic evaluation of the graph's edges was enabled according to the filters entered by the user, and an algorithm was created to return several routes that maximise compliance with those filters.

In parallel, a website prototype was produced, evaluated with real volunteers, and ultimately turned into a simple, intuitive and user-friendly application.

Nevertheless, the project has some limitations. First, the quality and availability of Google Maps reviews vary greatly between stations, so the reliability of the filters depends directly on the volume of opinions collected. Second, the number of filters available to the user is limited (only eleven). Finally, the application does not take into account dynamic variations in Madrid Metro service—such as temporary delays, line incidents or changes in train frequency—because it relies on static review data rather than real-time information.

7.2. Future Work

Looking ahead, several improvements are envisaged. Enhancing the application with more filters—or even allowing free-text filters—would be valuable. To increase its potential, users could be allowed to submit reviews directly through the website. This would provide continuous feedback and gradually enrich the database, improving the accuracy and timeliness of recommendations. With more reviews, the classifier could be retrained to filter new opinions more precisely.

At present, the graph representing the Madrid Metro network is undirected, assuming travel time between two stations is the same in both directions. One improvement would be to model the network as a directed graph, which would require adjusting the search algorithms.

Finally, on the web interface, it would be useful to overlay the routes returned by the algorithm on a map of Madrid. A real-time incident section for the metro network could also be added.

In short, this thesis has been a comprehensive exercise in computer engineering, encompassing data collection and processing, the design of AI models for text classification, the use of graphs for route calculation, and the design and validation of a user interface. 7.

Bibliografía

La ciencia no es más que una perversión de sí misma a menos que tenga como objetivo final mejorar la humanidad.

Nikola Tesla

et al., F. R. Css grid layout module level 2. 2024.

Alican Bozyiğit, G. A. y. E. N. Public transport route planning: Modified dijkstra's algorithm. 2017.

Areiza, S. A. Algoritmo para calcular la ruta más segura y óptima. 2019.

Contributors, H. Hunspell: A spell checker and morphological analyzer library. 2025.

Cursa. Herramientas de diseño de ux/ui. 2025.

Datasets, T. Tensorflow datasets catalog: C4 dataset. 2023.

EJS. Separación de palabras en python. 2016.

Face, H. Dataset hugging face. 2025a.

Face, H. Dataset hugging face process. 2025b.

Face, H. Hugging face transformers documentation: Gpt-2 model. 2025c.

Face, H. Hugging face transformers documentation: T5 model. 2025d.

FastApi. Fastapi: Modern, fast (high-performance), web framework for building apis with python. 2025.

GeeksforGeeks. enchant.dict_exists() in python. 2020.

GoogleTrans. Googletrans: Free and unlimited python library that implemented google translate api. 2023.

GoogleTranslator. Deep translator: A flexible free and unlimited python tool to translate between different languages. 2025.

graphlib. graphlib — graph library for python. 2025.

Hebbar, S. S. T5: Overview. 2023.

Håkon Wium Lie y Bert Bos. Css level 2 (revision 2) — the box model. 2011.

python igraph. The igraph software package for complex network research. 2025.

Jeff. Creating a paraphrase generator model using t5 and deploying on ainize. 2021.

Jr., T. A. Css custom properties for cascading variables module level 1. 2020.

van Kesteren, A., Denicola, D., Jägenstedt, P. y Pieters, S. Html living standard. 2024.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., tau Yih, W., Rocktaschel, T. y Riedel, S. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021.

Microsoft. Workshop: Interact with openai models – part 2 labs: System message. 2023.

Mozilla Developer Network. Css flexible box layout – guide. 2024a.

Mozilla Developer Network. Css grid layout – guide. 2024b.

Mozilla Developer Network. `box-sizing` — css property. 2024c.

Mozilla Developer Network. `link rel=ïcon"` — favicon and touch icons. 2024d.

Mozilla Developer Network. `scrollbar-gutter` — css property. 2024e.

Nair, A. Leveraging n-grams to extract context from text. 2021.

neptune.ai. Técnicas de aumento de datos. 2024.

Networkit. Networkit: A tool suite for large-scale complex network analysis. 2025.

Networkx. Networkx: Python software for the analysis of networks. 2025.

nltk. Natural language processing with python. 2025.

OpenAI. Gpt-2: 1.5b release. 2019.

Peña, V. Encapsulamiento de objetos en php. 2020.

Puente, O. Figma tutorial para principiantes | aprende diseño web ui de manera simple a través de figma. 2022.

RoKa. pyenchant for other languages. 2021.

Scikit-learn. Scikit-learn. 2025.

Sentence-Transformers. Sentence-bert: Sentence embeddings using siamese bert-networks. 2025.

SerpApi. Google search engine results api. 2024.

Sharma, N. Hugging face pre-trained models: Find the best one for your task. 2025.

spaCy. spacy: Industrial-strength natural language processing in python. 2025.

Stanza. Stanza: A python natural language processing toolkit for many human languages. 2025.

Tab Atkins Jr. y Elika J. Etemad. Css flexible box layout module level 1. 2021.

The PHP Documentation Group. `include`, `require`, `include_once` and `require_once`. 2024.

The PHP Group. curl functions — php manual. 2024a.

The PHP Group. Php data objects (pdo) — php manual. 2024b.

The PHP Group. `http_build_query()` — php manual. 2024c.

The PHP Group. `json_decode()` — php manual. 2024d.

graph tool. The graph-tool python library. 2025.

deep translator. Deep translator: A flexible free and unlimited python tool to translate between different languages. 2025.

Translators. Translators python libray. 2024.

Unsloth. Unsloth. 2025.

Velázquez, D. Como crear un diagrama de flujo de usuarios en figma - tutorial español 2023. 2023.

wooorm. dictionaries: Spanish dictionary. 2025.

WordNet. Wordnet: A lexical database for english. 2025.

Zhang, Y. y Yang, Y. Releasing paws and paws-x: Two new datasets to improve natural language understanding models. 2019.