UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Interfaz gráfica para interactuar con resultados de evaluación orientada a sistemas de recomendación

Ricardo Morato Mateos
Tutor: Alejandro Bellogín Kouki
Ponente: Iván Cantador Gutiérrez

MAYO 2018

Interfaz gráfica para interactuar con resultados de evaluación orientada a sistemas de recomendación

AUTOR: Ricardo Morato Mateos
TUTOR: Alejandro Bellogín Kouki

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2018

Resumen (castellano)

Se sabe que, en los últimos tiempos, con el uso masivo de internet, la importancia del análisis de los datos cada vez es mayor. Esto lo saben tanto investigadores como empresas, que cada vez recopilan más información de los usuarios para crear un perfil virtual de ellos. A través de estos perfiles, las empresas aportan sugerencias, que pueden ser productos de una tienda, artículos de un blog, etc. a los usuarios de forma precisa, con el uso de sistemas de recomendación.

El crear un buen sistema de recomendación puede suponer una gran ventaja a las empresas, tanto en ventas, haciendo que un usuario compre otro libro similar al que acabe de leer o descubra un video que le pueda gustar, como fidelizando al cliente, consiguiendo que estos vuelvan a su negocio.

Este Trabajo Fin de Grado se basa en el desarrollo de una aplicación para la evaluación de sistemas de recomendación. En ella se podrá tanto evaluar una ejecución, comparando distintas métricas y cutoffs, como comparar varias ejecuciones, de una forma visual y atractiva. La creación de una aplicación como esta puede ayudar a los desarrolladores de los sistemas de recomendación, ya que una buena evaluación de estos es crucial para saber cuán bueno es.

Para el desarrollo de este trabajo se ha realizado un estudio de las formas de evaluación de los sistemas de recomendación, así como de la elección de las tecnologías usadas. La aplicación se ha desarrollado en JavaScript, tanto en la parte *frontend*, con React.js, como en la parte *backend*, utilizando el entorno Node.js. Dado que el entorno JavaScript no es muy demandado en este tipo de proyectos, se ha tenido que desarrollar todo desde cero; para ello, la realización de una buena batería de pruebas ha sido esencial.

Palabras clave (castellano)

Sistemas de recomendación, evaluación, evaluación offline, métricas de evaluación, aplicación web.

Abstract (English)

It is well-known that, in the last times, with the massive use of Internet, the importance of data analysis keeps growing all the time. Both researchers and companies are well aware of this fact, since they keep collecting more data and information about users in order to create a virtual profile of them. From these profiles, companies make suggestions, that could be items from a store, articles in a blog and so on. Recommendation systems are used in order to define these suggestions.

Building a good recommendation system could provide many advantages for companies from the sales point of view, by suggesting a user to buy a book similar to the one they have just read or by discovering a new video they could like, but also from the loyalty point of view, by making them to come back again.

This Bachelor thesis is based on the development of an application for the evaluation of recommendation systems. This application would allow to evaluate recommendation executions, by comparing different metrics and cutoffs using a visual and attractive way. Building such application may help developers to know how effective a recommendation system is.

A study about how recommendation systems are evaluated and about the most used technologies has been carried out in order to develop this thesis. The developed application is built in JavaScript, where we have used React.js in the frontend and Node.js in the backend. Since the JavaScript environment is not highly demanded for this kind of projects, everything was built from scratch. Because of that, a test battery was essential to guarantee a proper functioning of the application.

Keywords

Recommender systems, evaluation, offline evaluation, evaluation metrics, web application.

Agradecimientos

A todos los que me han apoyado.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Con el aumento del uso de internet, la cantidad de opciones a las que se enfrentan los usuarios son prácticamente ilimitadas. Elegir qué película ver o qué libro comprar se puede convertir en una difícil tarea por esta razón. Para ayudar a las personas a tomar estas decisiones, surgen los sistemas de recomendación. Estos sistemas son un conjunto de herramientas que, en base a unos gustos de un usuario, aportan sugerencias que puedan serle útiles, ahorrándoles esfuerzo y tiempo, descartando los artículos que no les aportan valor. Es por ello que se han convertido en una parte importante del negocio de las empresas, ya que el desarrollar un buen sistema de recomendación supone una ventaja competitiva con otras empresas, tanto en ventas, haciendo que un usuario compre otro libro similar al que acabe de leer o descubra un video que le pueda gustar, como fidelizando al cliente, consiguiendo que estos vuelvan a su negocio.

Por estos motivos, el estudio de estas herramientas tiene cada vez más peso en la investigación. Cada vez se dedican más esfuerzos en la recopilación de datos y el análisis de estos, con investigadores y empresas dedicados íntegramente a esta labor.

El desarrollo de aplicaciones que ayuden al análisis de la información es siempre bienvenido y, en concreto, una aplicación como la desarrollada en este trabajo puede ser de gran ayuda, ya que no existe ninguna herramienta visual parecida para la evaluación de sistemas de recomendación, siendo esta parte muy importante, ya que una buena evaluación de estos sistemas es crucial para saber cuán bueno son.

1.2 Objetivos

El objetivo de este trabajo es desarrollar una aplicación web que facilite la evaluación de sistemas de recomendación para ayudar a los desarrolladores de estos. La aplicación le mostrará información al usuario, de forma textual y en forma de gráfico, sobre las distintas métricas de evaluación, y dentro de estas, sobre las distintas formas de ejecutarlas. El desarrollador podrá subir al sistema tanto conjuntos de datos (datasets) como ejecuciones de algoritmos de recomendación sobre dichos datasets, pudiendo así compararlas, para una fácil selección del mejor algoritmo, y un análisis de los distintos algoritmos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción.** Resumen sobre el porqué del desarrollo del proyecto y de los objetivos propuestos para este.

- **Estado del arte.** Estudio sobre el estado actual de los sistemas de recomendación y sus distintas formas de evaluación.
- **Estado de las tecnologías.** Estudio sobre las tecnologías utilizadas, así como de otras herramientas usadas en el desarrollo de la aplicación.
- **Diseño.** Análisis sobre los requisitos de la aplicación.
- **Desarrollo.** Explicación sobre el desarrollo de la aplicación.
- **Pruebas.** Explicación de las distintas pruebas a diferentes niveles realizadas sobre la aplicación para comprobar su correcto funcionamiento, así como los resultados de estas.
- **Conclusiones y trabajo futuro.** Resumen de los conocimientos adquiridos y posibles mejoras sobre el trabajo desarrollado.

2 Estado del arte

2.1 ¿Qué es un sistema de recomendación?

Un sistema de recomendación es un conjunto de herramientas y software que proporcionan sugerencias que sean de utilidad para un usuario. Para este propósito, los sistemas de recomendación se basan en diferentes algoritmos de predicción, como kNN, arboles de decisión o redes neuronales. Estos algoritmos procesan una información de entrada, generalmente puntuaciones dadas por los usuarios a los ítems del sistema (Tabla 1) y devuelven en su salida las recomendaciones, donde lo más habitual es generar un ranking de los ítems para cada usuario (Tabla 2).

Usuario	*Ítem*	*Valoración*
1	*1*	*1*
1	*2*	*3*
1	*3*	*2*
1	*4*	*3*
1	*5*	*4*

Tabla 2-1: Ejemplo de entrada de un sistema de recomendación.

Usuario	*Ítem*	*Valoración*
1	*7*	*4*
1	*8*	*4*
1	*2*	*3*
1	*3*	*3*
1	*10*	*3*
1	*1*	*2*
1	*5*	*2*
1	*4*	*1*
1	*6*	*0*
1	*9*	*0*

Tabla 2-2: Ejemplo de salida de un sistema de recomendación.

2.2 Tipos de sistemas de recomendación

Entre los varios tipos de sistemas de recomendación existentes, se pueden destacar dos como los más usados: los sistemas basados en contenido y los sistemas de filtrado colaborativo.

2.2.1 Sistemas basados en contenido

Recomiendan ítems asociados a los ítems que gustaron a un usuario en el pasado [12]. Para ello, se tiene en cuenta las características de estos ítems y se intenta predecir en base a ellas, buscando ítems que tengan unas características similares.

A continuación, se describen las ventajas y desventajas de estos sistemas [12]:

Ventajas

- Independencia del usuario: No hace falta de más usuarios para recomendar un ítem a un usuario.
- Transparencia: Se sabe qué características tienen los ítems que le gustan a un usuario.
- Nuevo artículo: Son capaces de recomendar ítems que aún no han sido calificados por ningún usuario.

Inconvenientes

- Análisis de contenido limitado: A veces un sistema de recomendación no predice bien si no tiene las suficientes características de los ítems.
- Sobre-entrenamiento: Es difícil que encuentre un ítem de una nueva categoría que le vaya a gustar al usuario, i.e., no va a recomendar películas de otro género que no haya puntuado el usuario.
- Nuevo usuario: Sin los suficientes ítems puntuados, el sistema no va a ser capaz de encontrar otro de forma precisa, i.e., para un nuevo usuario.
- Arranque en frio. Sin datos no pueden recomendar.

2.2.2 Sistemas de filtrado colaborativo

Recomiendan ítems en base a los gustos de otros usuarios similares y las características de los ítems que le gustaron en el pasado [10]. Hay varias formas de medir la similitud entre usuarios. Una de ellas es la similitud coseno [11]:

$$sim(A, B) = \cos(A, B)$$

donde
A = vector que representa al usuario A
B = vector que representa al usuario B

A continuación, se describen las ventajas y desventajas de estos sistemas [3]:

Ventajas

- Ítems de otras categorías: Son capaces de recomendar ítems de categorías que no haya clasificado el usuario al basarse en usuario similares.

- Mejor evaluación del contenido: Evalúan la calidad de los ítems con puntuaciones de otros usuarios, una medida más realista que evaluar los ítems en base a su contenido, ya que puede ser incompleto y no tener todas las características.
- Cambio en los intereses del usuario: Al cambiar los intereses de los usuarios, los ítems a recomendar deben ser distintos y estos sistemas se adaptan a ese cambio.

Inconvenientes

- Nuevo artículo: Los ítems deben tener puntuaciones para que puedan ser recomendados.
- Oveja gris: Un usuario deber ser similar a otros para recibir recomendaciones, si no, será difícil que reciba sugerencias útiles.
- Nuevo usuario: Sin los suficientes ítems puntuados, el sistema no va a ser capaz de encontrar otro de forma precisa, i.e., para un nuevo usuario.
- Arranque en frio. Sin datos no pueden recomendar.

2.3 Algoritmos de predicción

Los algoritmos de predicción se dividen en dos tipos: supervisados, en el cual se sabe la clase antes de ejecutarlos y no supervisados, en los que no se sabe la clase [2]. A continuación, se describen tres de entre los muchos algoritmos existentes [2].

2.3.1 kNN

Es uno de los algoritmos más sencillos de aprendizaje automático y uno de los más usados para los sistemas de filtrado colaborativo [2]. kNN calcula la distancia de un ítem o un usuario sobre los demás y escoge los *k* más cercanas. Para tener un buen sistema de recomendaciones es importante la elección de *k*. Si *k* es pequeño, es posible que buenas recomendaciones se queden sin ser seleccionadas y si *k* es grande las recomendaciones se pueden mezclar con ruido de vecinos no tan cercanos. kNN no construye un modelo, sino que calcula las distancias cada vez que se quiera clasificar, por ello, es más costoso computacionalmente que otros algoritmos.

2.3.2 Árboles de decisión

Crean un modelo en base a un diagrama con forma de árbol. Estos árboles dividen el espacio usando unas reglas. Se componen de un nodo raíz del cual se parte, nodos de decisión, en los que se toma una decisión en base a unas reglas aplicadas sobre uno o varios atributos y nodos hojas, que indican el valor de la clase que se escogerá en caso de llegar a ellos.

Figura 2-1: Árbol de decisión.

2.3.3 Redes neuronales

Una red neuronal es un conjunto de nodos interconectados y enlaces ponderados que se inspira en la arquitectura del cerebro biológico [2]. Estos nodos son llamados neuronas. Cada uno de estos nodos tiene una función que devuelve un resultado dependiendo de la entrada. Por último, la capa de salida tiene una función de 'activación', que puede ser binaria (la neurona se activará o no) o no.

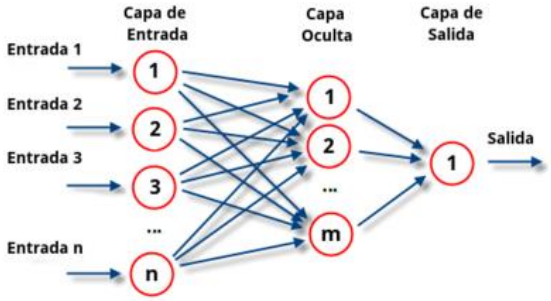

Figura 2-2: Red neuronal.

2.3.4 Factorización de matrices

También llamados modelos basados en SVD (Singular Value decomposition) [10], estos algoritmos representan tanto a los usuarios como a los ítems en forma de vector. En su forma básica, SVD, la predicción del rating se obtiene realizando el producto de ambos [10].

A partir del modelo básico, surgen variaciones, como SVD++, que tiene en cuenta el feedback implícito, como puede ser el número de comentarios o el número de artículos valorados, sin contar su puntuación [10].

2.4 Evaluación de los sistemas de recomendación

Hay dos formas de evaluar un sistema de recomendación: de forma online, en la cual se necesita la implicación de los usuarios para establecer una medida de la calidad del sistema [4] y de forma offline, con diferentes métricas basadas en error o en precisión. Este trabajo se ha centrado en la evaluación offline.

2.4.1 Métricas basadas en error

Son métodos estadísticos que se basan en el error para saber si un sistema de recomendación es bueno o no. Es decir, a más error, peores recomendaciones. Para el cálculo de este error se compara la recomendación real frente a la recomendación predicha.

El principal inconveniente de estas métricas es el no hacer distinciones entre los ítems principales y los secundarios; dan el mismo peso a la recomendación décima que a la primera [4]. A continuación, se presentan las métricas principales basadas en error.

MAE

Mide la desviación de las recomendaciones respecto a su valor real. Esta métrica trata con el error absoluto [14].

$$MAE = \frac{1}{Te} \sum_{(u,i) \in Te} |\hat{r}(u,i) - r(u,i)|$$

siendo
Te = Conjunto de usuarios y recomendaciones
r̂(u,i) = Recomendación predicha i del usuario u
r(u,i) = Recomendación real i del usuario u

RMSE

Al igual que MAE, mide la desviación de las recomendaciones respecto a su valor real. Al elevar el error al cuadrado, RMSE penaliza más los errores altos. Es una de las métricas más utilizadas [1].

$$RMSE = \sqrt{\frac{1}{Te} \sum_{(u,i) \in Te} (\hat{r}(u,i) - r(u,i))^2}$$

siendo
Te = Conjunto de usuarios y recomendaciones
r̂ (u,i) = Recomendación predicha i del usuario u
r(u,i) = Recomendación real i del usuario u

2.4.2 Métricas basadas en la precisión

Cuando se desea medir la cantidad de elementos recuperados relevantes y no relevantes, en lugar de métricas basadas en error se utilizan métricas basadas en precisión [5]. Generalmente, para el cálculo de estas métricas es necesario traducir la escala de clasificación a binario (si no lo está ya), i.e., en el conjunto de datos de Movielens [20], la escala de calificaciones está entre 1 y 5. Normalmente, se suele considerar relevantes las calificaciones entre 4 y 5. El resto se considerarían no relevantes. Para la traducción a binario las calificaciones entre 4 y 5 tomarían el valor *relevante* y el resto el valor *no relevante*. La excepción sería nDCG, en la cual la relevancia puede ser o binaria o la escala real.

Precisión y Recall

Después de la traducción a binario de la escala de clasificación, es necesario calcular la matriz de confusión [7]. Esta matriz muestra los valores que puede tomar cualquier recomendación.

	Seleccionado	No seleccionado
Relevante	RS	NRS
Irrelevante	IS	NIN

Tabla 2-3: Matriz de confusión para el cálculo de precision y recall

A continuación, se indica cómo se calculan las métricas de precisión y recall según los valores de la matriz de confusión. Antes de eso, es importante notar que estas dos métricas están relacionadas: mejorar el recall hará que la precisión disminuya y viceversa

Precisión

Muestra la capacidad del sistema de recomendación para mostrar sólo elementos útiles [7].

$$P = \frac{RS}{RS + NRS}$$

Esta métrica se suele calcular con un cutoff y se llama P@k. Por ejemplo, si se hiciesen diez recomendaciones para cada usuario y quisiéramos medir la precisión hasta la recomendación tres (P@3), se evaluarían únicamente las tres primeras recomendaciones de cada usuario.

Recall

Probabilidad de que un ítem relevante sea recomendado [1]. También se puede describir como la probabilidad de que un documento relevante, seleccionado al azar, sea recuperado en una búsqueda.

$$R = \frac{RS}{RS + IS}$$

Al igual que la precisión, recall suele calcularse con un cutoff. Se llama R@k.

F

También llamada media armónica [9], combina precisión y recall en una sola métrica. Es usada a menudo como un indicador del rendimiento general del sistema de recomendación [15]. Cuanto más cerca de uno, mejores predicciones.

$$F = \frac{2PR}{P + R}$$

siendo *P = Precisión y R = Recall*

MRR

Indica en qué rango está el primer ítem relevante, de media, para los usuarios [13]. Favorece las clasificaciones cuyo primer resultado correcto ocurre cerca de los mejores resultados de clasificación.

$$MRR = \frac{1}{U} \sum_{u \in U} \frac{1}{rank(u)}$$

siendo
U = Conjunto de usuarios
rank(u) = Posición del primer elemento bien recomendado

MAP

Calcula el promedio de las precisiones de todos los ítems relevantes para todos los usuarios.

$$MAP = \frac{1}{U} \left(\sum_{u \in U} \left(\sum_{k \in K} \frac{P@k * rel(k)}{total\ relevantes} \right) \right)$$

siendo
U = Conjunto de usuarios
K = Cutoff
P@k = Precisión hasta el punto k
rel(k) = Si el ítem k es relevante
total relevantes = Número total de ítems relevantes

nDCG

nDCG mide la utilidad de un ítem basado en su posición en una lista de resultados [6]. Esta métrica penaliza si un ítem relevante aparece en una parte baja del ranking. Puede tratar la relevancia de forma numérica o binaria.

Para su cálculo, es necesario calcular previamente DCG e iDCG, donde iDCG es el DCG ideal, es decir, si todas las recomendaciones relevantes estuvieran al inicio del ranking, y DCG es el valor asociado a una lista de recomendación.

$$nDCG = \frac{1}{U}\sum_{u \in U} \frac{DCG(u)}{iDCG(u)}$$

$$DCG = \sum_{i \in u} \frac{2^{rel(i)}}{log_2(1+i)}$$

siendo

rel(i) = Relevancia de la recomendación i

3 Estudio de las tecnologías a utilizar

La aplicación se desarrollará bajo un patrón MVC, como se explica en el punto 5.1. Esta sección se ha divido en base a lo propuesto para este patrón.

3.1 Vista

HTML

Es un lenguaje de marcado con el cual se puede estructurar el contenido de una web. Es básico para la creación de este tipo de aplicaciones. Actualmente se encuentra en la versión 5, añadiendo nuevas etiquetas como *nav*, *section* y *footer*, las cuales permiten un mayor sentido semántico a la estructura de la aplicación.

CSS

Es el lenguaje para el diseño visual de una aplicación web. Se aplica sobre los elementos HTML modificando su apariencia (tamaño, posición, color, etc.).

JavaScript

JavaScript es un lenguaje de programación basado en eventos y que permite añadir interactividad a la parte front-end de una aplicación web.

En los últimos años se ha cambiado la forma de desarrollar la parte front-end de una aplicación gracias a la llegada de frameworks o librerías para JavaScript. Una de las más conocidas y que se ha utilizado para desarrollar la vista de la aplicación es React [28].

React es una librería desarrollada por Facebook. Utiliza un DOM virtual que detecta los cambios y los aplica sobre el DOM real, aumentado así la velocidad de renderizado. Para ello, crea un árbol a partir del DOM y así sólo tiene que renderizar de nuevo los nodos que han cambiado. Cada componente maneja un estado y cuando cambia ese estado, se vuelve a renderizar el componente. React es muy ligero, ocupando únicamente 132 Kb, aumentando así la velocidad de carga de las webs que lo utilizan, ya que frameworks como Angular ocupan más de 500 Kbs.

Para aumentar su funcionalidad se pueden añadir librerías, i.e., React-router, para manejar el enrutamiento y poder crear aplicaciones de una sola página (SPA) o Redux, para introducir un patrón de diseño Flux.

Al hacer uso de funcionalidades introducidas en la versión ES6, es necesario utilizar un compilador, para que traduzca el código de la versión ES6 a la versión ES5 y pueda ser usada en navegadores antiguos, que utilicen la versión ES5 de JavaScript. El compilador más utilizado es Webpack [33]

Empresas grandes están desarrollando sus aplicaciones con React, empresas como [32]: Facebook, Instagram, Microsoft, Netflix, Airbnb, New York Times, Yahoo, Whatsapp, Codeacademy, Dropbox o Atlassian.

3.2 Controlador

3.2.1 Tecnología escogida

Se ha elegido el uso de JavaScript en el lado servidor y para ejecutarlo, es necesario apoyarse en Node.js.

Node.js es un entorno basado en el motor V8 de Google. Su ejecución es monohilo y con llamadas a funciones E/S de forma asíncrona, ahorrando el cambio de contexto y su coste. A partir de la versión ES6 de JavaScript existe la posibilidad de hacer llamadas síncronas con las sentencias *async* y *await*. Por otro lado, actualmente existen librerías para poder ejecutar múltiples hilos con Node, como Napa.js, desarrollada por Microsoft [25]. Tiene su propio gestor de paquetes, npm, para la instalación y actualización de librerías.

Como punto negativo, hay escasas librerías para el aprendizaje automático y ninguna completa para la evaluación de sistemas de recomendación. Por ello, se decidió desarrollar una librería con ese uso, descrita en el apartado 4.3.4.

Aun sabiendo este problema, se ha escogido JavaScript por dos razones: La primera, la necesidad de compilar la parte frontend de la aplicación. Esto es mandatorio hacerlo desde Node.js y si se quiere usar otro lenguaje para el controlador, será necesario tener otro servidor que interprete ese lenguaje, teniendo dos servidores para la misma aplicación. Esto retrasaría y complicaría el desarrollo, teniendo que configurar dos servidores y desarrollar en dos lenguajes distintos. La segunda razón es el interés del autor de este trabajo por aprender esta tecnología.

3.2.2 Otras tecnologías

Java

Lenguaje ampliamente utilizado, tiene entre sus características principales el ser un lenguaje orientado a objetos, fuertemente tipado, multihilo y portable, gracias a la utilización de una máquina virtual, JVM, capaz de interpretar el lenguaje en cualquier máquina.

La principal ventaja que tiene Java para el desarrollo de esta aplicación sería el tener una librería como RiVal [31], que es un conjunto de herramientas entre las que se incluyen la evaluación de sistemas de recomendación.

PHP

El lenguaje por excelencia del desarrollo web. Es utilizado por empresas como Facebook o Wordpress. Tiene una gran comunidad detrás y la documentación es una de las más completas que se pueden encontrar, estando traducida a diferentes idiomas. Cuenta con varios frameworks MVC para la simplificación del desarrollo, como Symfony o Laravel. Es un lenguaje multiplataforma y permite el desarrollo orientado a objetos.

3.3 Modelo

La realización del modelo se ha desarrollado con una base de datos no relacional. Estas bases de datos se caracterizan por ser más rápidas que las bases de datos relaciones (SQL) al no tener relaciones entre las tablas, por crecer de forma horizontal en vez de en vertical y por no tener operaciones atómicas.

Existen varios tipos de bases de datos no relacionales: orientadas a documentos, como MongoDB, de tipo clave – valor, como Redis, etc.

En este proyecto se ha usado MongoDB, por la facilidad de integración con Node.js, utilizando la librería Mongoose [24].

MongoDB es una base de datos orientada a documentos, en la que los datos son guardados en objetos JSON en las tablas. Un documento de MongoDB sería como una fila en una base de datos relacional, pero con un gran cambio, los documentos no tienen por qué tener los mismos campos. MongoDB permite realizar operaciones de búsqueda por campos, aplicando filtros o expresiones regulares sobre ellos, cosa que, por ejemplo, las bases de datos clave - valor no permiten.

Mongoose es una librería que permite definir los modelos de forma sencilla, aplicar restricciones en sus campos y exportarlos para su uso en los controladores, dando acceso a operaciones CRUD sobre estos.

4 Diseño

4.1 Análisis de requisitos

4.1.1 Requisitos funcionales

RF1 **–** Almacenamiento de datasets: el usuario podrá subir todos los datasets que desee.

RF1.1 – Cada dataset deberá contener la siguiente información asociada:

- Nombre: Deberá ser único y sólo puede contener caracteres alfanuméricos y los caracteres '.' y '_'.
- Descripción: Pequeña descripción para que el usuario tenga conocimiento del contenido del dataset.
- Fichero de entrenamiento: Fichero con el cual se entrenó al modelo en el cual se basan las recomendaciones.
- Fichero de test: Fichero con el cual se probó al modelo en el cual se basan las recomendaciones.
- Fecha de subida: Fecha en la que se almacenó el dataset en la aplicación. Se calculará automáticamente.
- Atributos de los ficheros: Nombre de los atributos que contienen los ficheros de entrenamiento y test.
- El atributo que contiene la puntuación.
- El atributo que contiene el ítem que se ha puntuado.
- El atributo que contiene el usuario que ha puntuado.
- A partir de qué puntuación se considera relevante un ítem.

RF2 – Consulta de datasets: El usuario podrá consultar cada uno de los datasets almacenados.

RF2.1 – Se visualizarán las últimas ejecuciones asociadas al dataset, con un máximo de cuatro.

RF2.2 – Visualización de los campos descritos en el requisito funcional RF1.1

RF2.3 – Visualización del número de veces que se ha consultado cada métrica ejecutada sobre el dataset.

RF2.4 – Características del fichero de entrenamiento

RF2.4.1 – Visualización del número de entradas del fichero.

RF2.4.2 – Visualización del número de repeticiones de los valores del atributo de rating.

RF2.4.3 – Visualización del número de ítems relevantes y no relevantes.

RF2.5 – Características del fichero de test

RF2.5.1 – Visualización del número de entradas del fichero.

RF2.5.2 – Visualización del número de repeticiones de los valores del atributo de rating.

RF2.5.3 – Visualización del número de ítems relevantes.

RF3 – Búsqueda de datasets: Para un acceso rápido, el usuario podrá realizar una búsqueda sobre los distintos datasets.

RF4 **–** Borrado de datasets

RF5 **–** Subida de ejecuciones con recomendaciones.

RF5.1 – Cada ejecución deberá contener la siguiente información asociad:

- Nombre: Deberá ser único y sólo puede contener caracteres alfanuméricos y los caracteres '.' y '_'.
- Descripción: Pequeña descripción para que el usuario tenga conocimiento del contenido del dataset.
- Fichero de recomendaciones: Fichero con el cual se encuentran las recomendaciones dadas por el modelo.
- Fecha de subida: Fecha en la que se almacenó la ejecución en la aplicación. Se calculará automáticamente.
- Atributos de los ficheros: Se obtendrá del dataset asociado.
- El atributo que contiene la puntuación: Se obtendrá del dataset asociado.
- El atributo que contiene el ítem que se ha puntuado: Se obtendrá del dataset asociado.
- El atributo que contiene el usuario que ha puntuado: Se obtendrá del dataset asociado.
- A partir de qué puntuación se considera relevante un ítem: Se obtendrá del dataset asociado.

RF6 **–** Borrado de ejecuciones

RF7 – Consulta de ejecuciones

RF7.1 – Se podrá consultar las métricas que se elijan de entre las descritas en el estado del arte.

RF7.2 – Se podrá aplicar uno o varios cutoffs a las métricas escogidas.

RF7.3 – Se podrá filtrar los datos por al menos uno de los atributos.

RF7.4 – Se visualizarán gráficas comparativas sobre las distintas métricas seleccionadas y los distintos cutoffs.

RF8 – Comparación de ejecuciones

RF8.1 – Se podrá consultar las métricas que se elijan de entre las descritas en el estado del arte.

RF8.2 – Se podrá aplicar uno o varios cutoffs a las métricas escogidas.

RF8.3 – Se visualizarán gráficas comparativas sobre las distintas métricas seleccionadas y los distintos cutoffs.

RF9 – Creación de una página principal.

RF9.1 – Se visualizarán los últimos datasets almacenados con un máximo de cuatro.

RF9.2 – Se visualizará el número total de datasets y el número total de ejecuciones.

RF9.3 – Visualización del número de veces que se ha consultado cada métrica.

RF10 – Creación de una página que contendrá todos los datasets almacenados, ordenados por fecha de creación.

RF11 – Creación de una página que contendrá todas las ejecuciones almacenadas sobre un dataset, ordenadas por fecha de creación.

RF12 – Información sobre las métricas: Se mostrará una pequeña descripción sobre las métricas, la cual contendrá el objetivo de esta, y su fórmula.

4.1.2 Requisitos no funcionales

RNF1 – Diseño moderno y sencillo: Para ello, la aplicación tiene que seguir el patrón de diseño Material Design [21].

RNF2 – Diseño intuitivo: El usuario debe poder navegar por la aplicación sin necesidad de la creación de un manual de usuario.

RNF3 – Rapidez: La aplicación deberá responder lo más rápido posible. No se establece un mínimo de tiempo en cada ejecución y/o cálculo, ya que el tamaño de los datasets y/o las ejecuciones, en las cuales se basan los cálculos, es muy variable y pueden ir desde unas pocas líneas hasta millones. Por esto, se deja en manos del desarrollador la optimización de las consultas.

4.2 Casos de uso

4.2.1 Casos uso sobre el objeto *dataset*

Este diagrama muestra las acciones que podrá realizar un usuario con respecto a los objetos *dataset*. Como se puede ver en la Figura 4-1, un usuario puede subir, consultar o borrar un dataset.

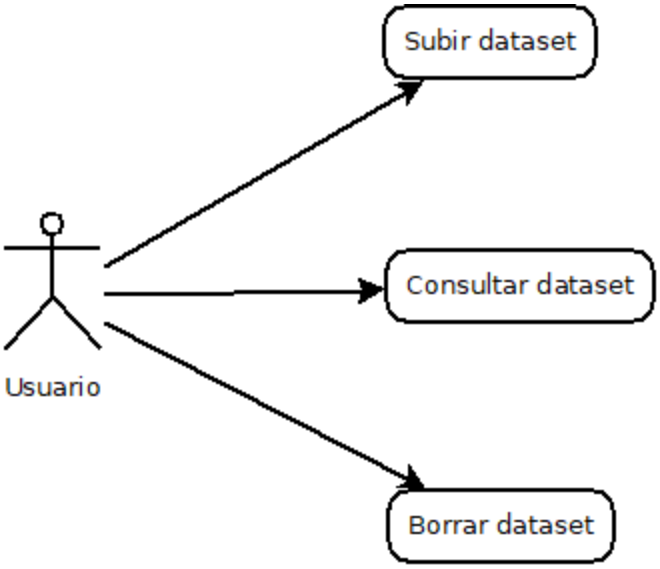

Figura 4-1: Casos de uso sobre el objeto *dataset*.

4.2.2 Caso de uso sobre el objeto *ejecución*

Este diagrama muestra las acciones que podrá realizar un usuario con respecto a los objetos *ejecución*. Un usuario puede subir, consultar o borrar una ejecución. También podrá comparar ejecuciones. En la consulta y comparación de ejecuciones, podrá consultar distintas métricas con distintos cutoffs. En la consulta podrá, además, ejecutar un filtro para los datos.

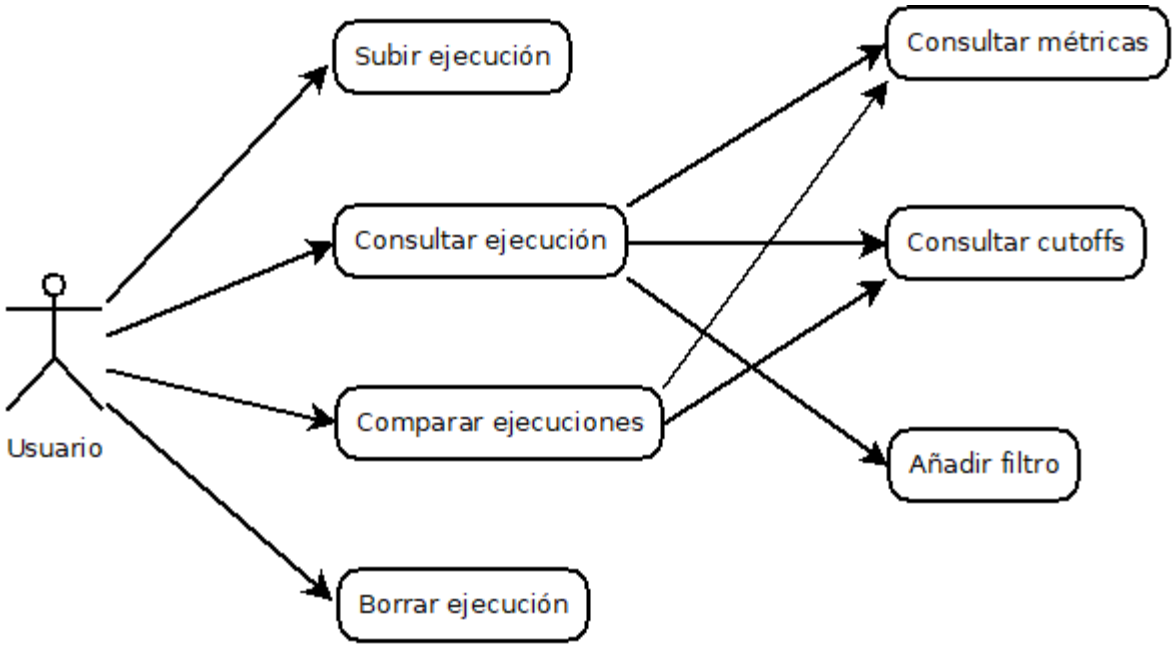

Figura 4-2: Casos de uso sobre el objeto ejecución.

5 Desarrollo

5.1 Patrón de diseño

Para el desarrollo de la aplicación se ha decidido usar el patrón MVC (Modelo – Vista - Controlador). Permite abstraer cada parte de la aplicación y que, al realizar un cambio en alguna de ellas, no afecte a las demás.

Este patrón separara la aplicación en tres partes:

- Vista: Se encarga de la representación de los datos y de las interacciones del usuario.
- Controlador: Se encarga de realizar la lógica de negocio.
- Modelo: Se encarga del acceso, recuperación y guardado de los datos en la base de datos.

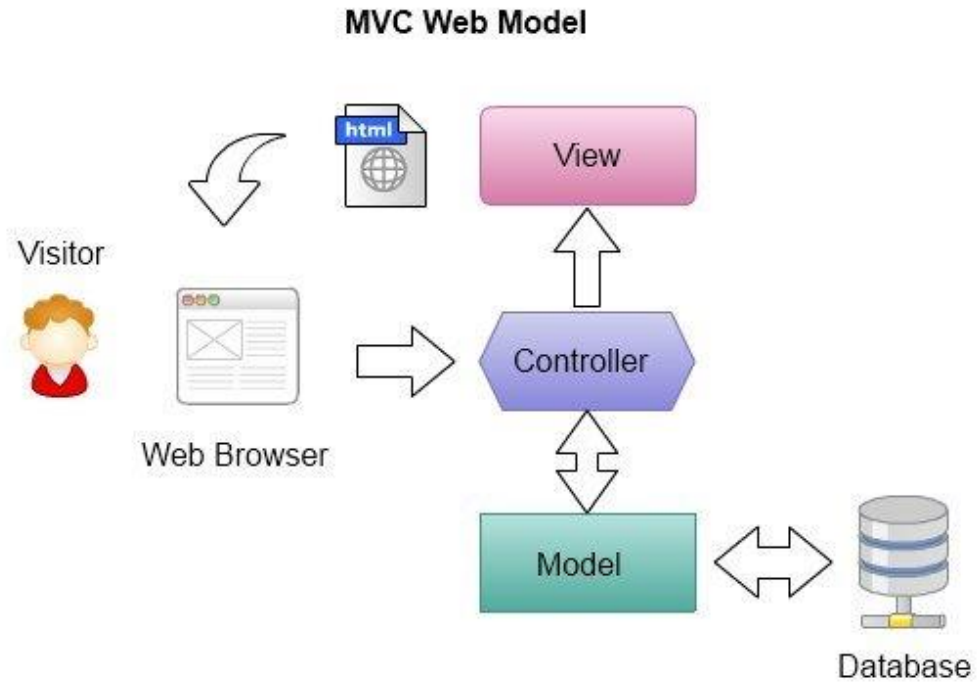

Figura 5-1: Patrón Modelo Vista Controlador.

Como se puede ver en la Figura 5-1, el usuario interacciona con el controlador a través del navegador. El controlador realiza operaciones sobre el modelo, que a su vez las

replica en la base de datos. Una vez realizadas, envía los datos a la vista, que será la encargada de mostrar los datos en el navegador.

5.2 Desarrollo

5.2.1 Modelo

El uso de Mongoose permite que los modelos sean declarados de una forma sencilla. Simplemente hay que crear un objeto de tipo *Schema*, describiendo los campos que tendrá el modelo, su tipo y sus restricciones. Esto nos da acceso a operaciones como el guardado o búsqueda con sólo importar el modelo, como se ve en la Figura 5-3.

```
const DatasetSchema = new Schema({
    name: {
        type: String,
        required: [true, 'Empty name']
    },
    description: {
        type: String,
        required: [true, 'Empty description']
    },
    train_file: {
        type: String,
        required: [true, 'Empty file']
    },
    test_file: {
        type: String,
        required: [true, 'Empty file']
    },
    timestamp: {
        type : Date,
        default: Date.now
    }
})
```

Figura 5-2: Definición del modelo *Dataset*.

```
const find = {name: name};
let dts = await Dataset.findOne(find, (err, dts) => {
    if (err) throw err;

    return dts
});
```

Figura 5-3: Uso del modelo para buscar un dataset por el campo nombre.

5.2.2 Vista

Se ha dividido toda la vista de la aplicación en componentes. Cada una de las pantallas tiene un componente *container*, que será el que incluya toda la lógica para mostrar en pantalla (llamadas a la API, formularios, etc.). Si la pantalla tiene elementos que se utilizan varias veces, se crea otro componente.

A continuación, en las Figuras 5-4 y 5-5, se puede ver un ejemplo de la pantalla que muestra una ejecución (*RunContainer.jsx*) y sus dependencias con otros componentes (*Table.jsx y MetricInfo.jsx*).

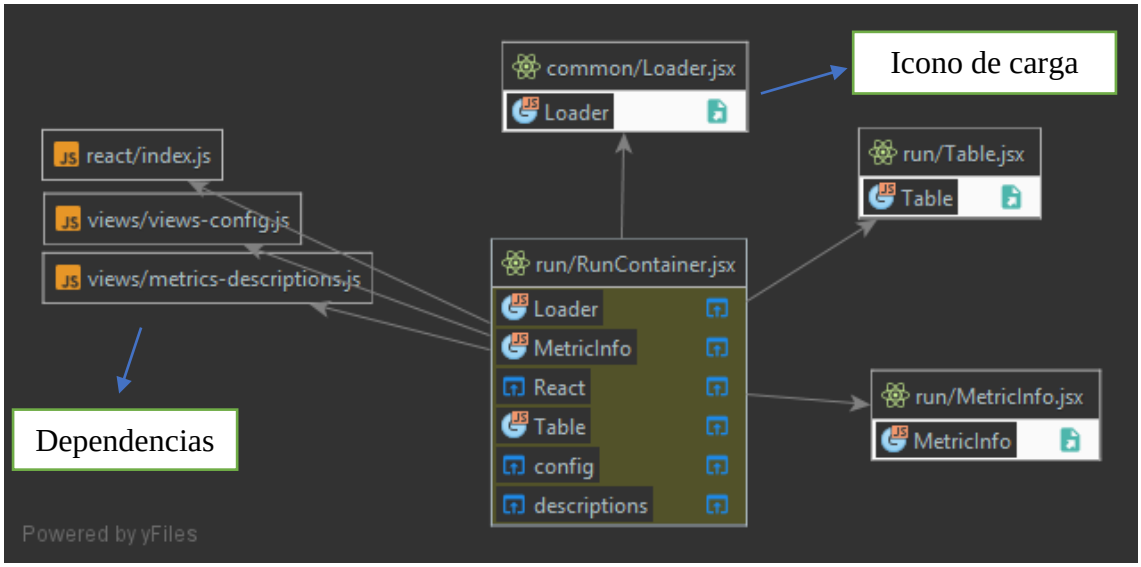

Figura 5-4: **Esquema de la pantalla que muestra una ejecución.**

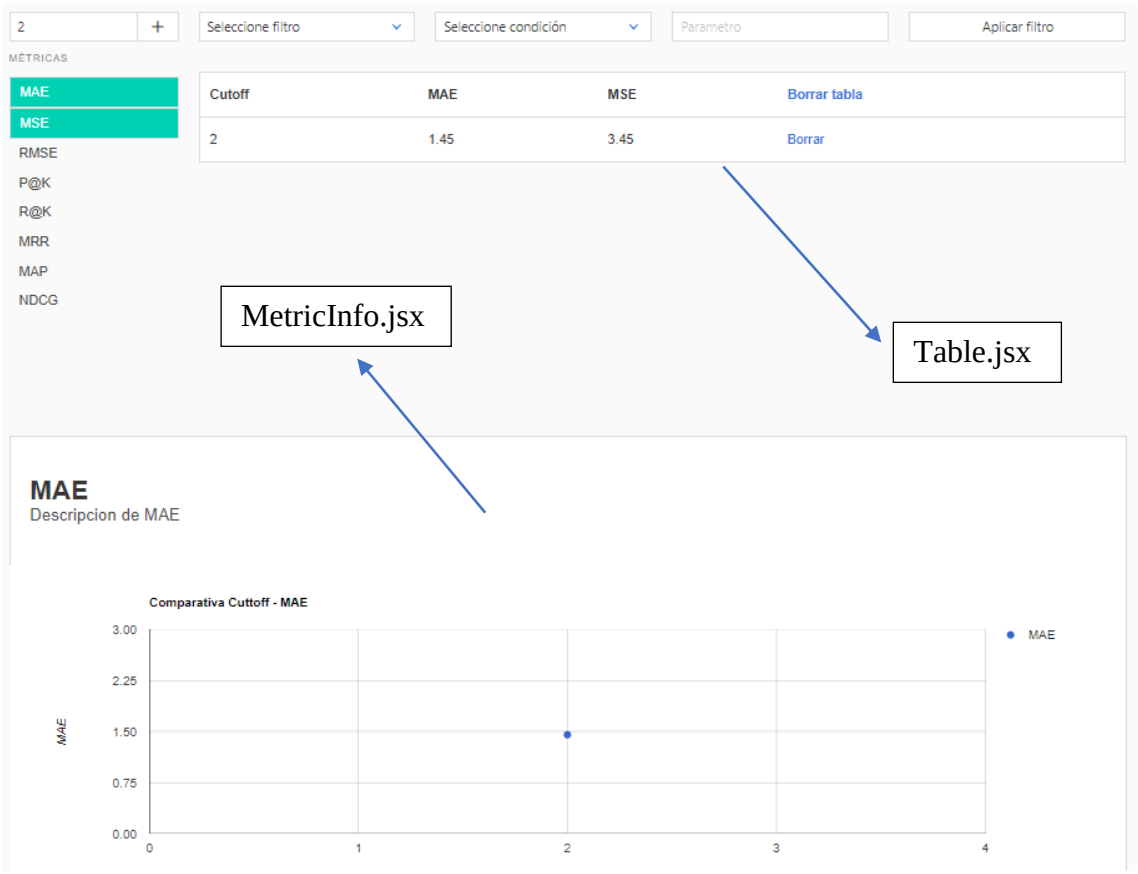

Figura 5-5: Pantalla que muestra una ejecución.

Para el diseño visual, se ha utilizado la librería Bulma [16], con algunos pequeños cambios. Esta es una librería CSS en la cual no se utiliza jQuery, cosa que otras más conocidas, como Bootstrap, sí hacen. El uso de jQuery podría llegar a suponer un problema, ya que tanto React como jQuery manejan el DOM de diferentes formas y dado que no se sabe qué código jQuery ejecutan estas librerías, se decidió tomar la alternativa de Bulma. Otra de las ventajas de Bulma es su ligereza, ocupando sólo 73 kb en su versión comprimida.

Con respecto a los gráficos de la aplicación, estos se han implementado con la librería Google Charts [19], usando la librería React Google Charts [29], que es una envoltura para usar la librería de Google con React.

Google Charts es una librería gratuita, con una gran variedad de gráficos. Es muy personalizable y permite la actualización de los gráficos en tiempo real, es decir, no tiene que volver a crear el gráfico cada vez se les inyecten nuevos datos.

5.2.3 Controlador

Se han tenido en cuenta unas consideraciones generales en el desarrollo de todos los endpoints del API:

1. Devolución de los datos en formato JSON.

2. Se han definido unos errores genéricos que devolverá el API si no se ha ejecutado bien la petición:

    - Error del servidor

        - Descripción: Ocurre un error inesperado en la ejecución del programa.
        - Código: 503
        - Mensaje: Internal server error.

    - Recurso ya existe

        - Descripción: Al subir un dataset o una ejecución, estos ya existen en la base de datos.
        - Código: 403
        - Mensaje: Resource already exists.

    - Recurso no encontrado

        - Descripción: Al consultar un dataset o una ejecución, estos no existen en la base de datos.
        - Código: 404
        - Mensaje: Resource doesn’t exists.

    - Parámetros incorrectos

        - Descripción: La petición recibe unos parámetros mal formados o que no puede procesar.
        - Código: 403
        - Mensaje: Bad params.

A continuación, se describen los endpoints desarrollados.

1. Endpoint */api/home*

    - Acción: GET
    - Descripción: Devuelve datos para presentar en la pantalla inicial.

2. Endpoint */api/datasets*

    - Acción: GET
    - Descripción: Devuelve un listado con todos los datasets.

3. Endpoint */api/datasets/:name*

    - Acción: GET
    - Descripción: Devuelve el dasatset especificado en name.

4. Endpoint */api/datasets*

    - Acción: POST
    - Descripción: Inserta un dataset en la base de datos.

5. Endpoint */api/datasets/:name*

    - Acción: DELETE
    - Descripción: Elimina el dasatset especificado en name.

6. Endpoint */api/datasets/:dataset_name/runs*

    - Acción: GET
    - Descripción: Devuelve un listado con todas las ejecuciones del dataset especificado en dataset_name.

7. Endpoint */api/datasets/:dataset_name/runs/:name*

    - Acción: GET
    - Descripción: Devuelve la ejecución especificada en name.
    - Parámetros:
        - Metrics: métricas a consultar.
        - Cutoff
        - Filter: filtro a aplicar sobre la ejecución.

8. Endpoint */api/datasets/:dataset_name/runs*

    - Acción: POST
    - Descripción: Inserta una ejecución en la base de datos.

9. Endpoint */api/datasets/:dataset_name/runs/:name*

    - Acción: DELETE
    - Descripción: Elimina la ejecución especificada en name.

10. Endpoint */api/datasets/:dataset_name/runs/:name/metrics*

    - Acción: GET
    - Descripción: Devuelve los resultados de las métricas que recibe por parámetro sobre la ejecución especificada en name.

11. Endpoint */api/datasets/:dataset_name/compare_runs*
    - Acción: GET
    - Descripción:
    - Parámetros:
        - Runs: ejecuciones que se van a comparar.
        - Metrics: métricas a consultar.
        - Cutoff

5.2.4 Librería para el cálculo de las métricas

Se ha desarrollado una librería que contiene todas las métricas descritas en el estado del arte. Como esta librería realiza muchas operaciones con matrices, ya que los ficheros a partir de los cuales se calculan las métricas pasan a ser matrices de datos, se decidió desarrollar una librería auxiliar para este propósito.

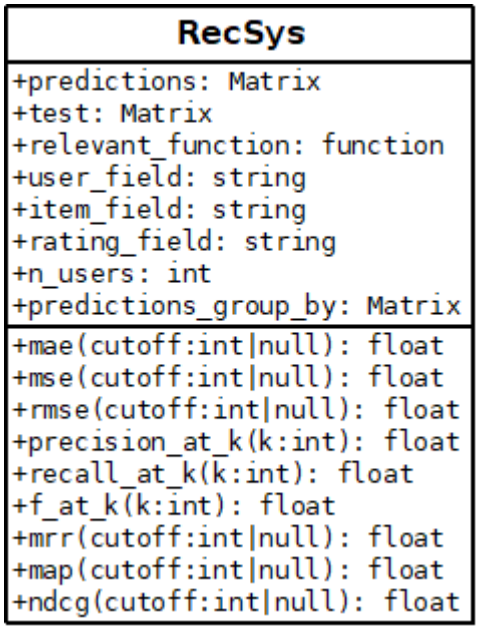

Figura 5-6: Diagrama de la librería RecSys [30]

5.2.5 Librería para el manejo de matrices

Esta librería contiene métodos (ver Figura 5-7) para el filtrado de datos que se aplican sobre una matriz. Por ejemplo, el método *select* devolverá la matriz sólo con los campos que se le pasen por parámetro. El método *where* filtrará los datos según unas condiciones que se le pasen por parámetro. Todos los métodos devuelven un objeto *Matrix*, para así poder encadenar llamadas, i.e., *matrix.select(...).where(...).orderBy*(…), devolverá un objeto *Matrix* con los campos pedidos en el método *select*, filtrando por unas condiciones que se aplican en el método *where* y ordenadas por la condición que se introduzca en *orderBy*.

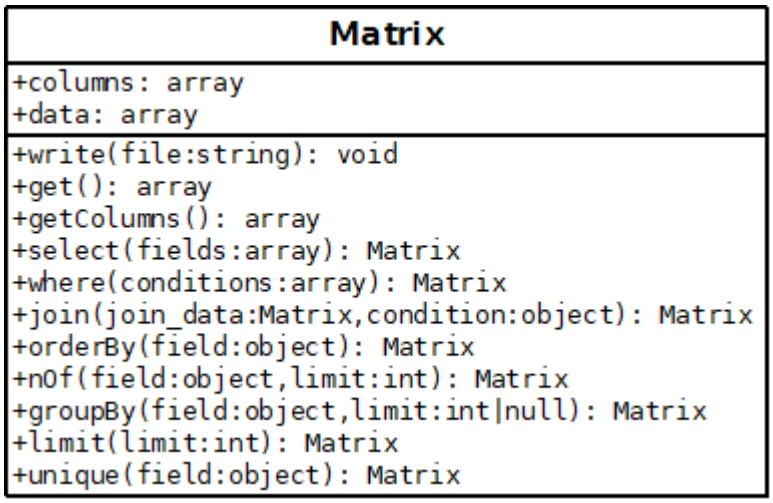

Figura 5-7: Diagrama de la librería Matrix [22]

5.2.6 Mejoras para aumentar la rapidez

Una vez desarrollada la aplicación, se vio que había un problema de rendimiento en el cálculo de las métricas. Las métricas tienen un buen rendimiento, pero había un problema en la instanciación de la librería que las calcula, tardando unos siete segundos en inicializarse con una ejecución de 300.000 filas. Por esto, se decidió aplicar el patrón de diseño *Container* [18].

Este patrón permite tener un contenedor en el cual se instancia el objeto que calcula las métricas y esté accesible cada vez que se quiera calcular una métrica, aunque sea en distintas peticiones.

También se decidió crear una tabla en la base de datos, llamada *cache*, en la que se introducía cada una de las métricas una vez estaban calculadas, para que la próxima vez que se consultasen, se obtuviesen de base de datos en vez de volver a calcularlas.

5.3 Diseño de la base de datos

En este apartado se muestran las tablas creadas para el almacenamiento de datos.

5.3.1 Tabla *datasets*

Esta tabla contendrá los atributos básicos de un dataset.

Clave	Name	description	train_file	test_file	timestamp
Tipo	string	string	string	string	date

Tabla 5-1: Tabla datasets.

5.3.2 Tabla *runs*

Esta tabla contendrá los atributos básicos de una ejecución.

Clave	name	description	file	dataset_name	timestamp
Tipo	string	string	string	string	date

Tabla 5-2: Tabla runs.

5.3.3 Tabla *files*

Esta tabla contendrá los atributos que compartirán los datasets y las ejecuciones.

Clave	*dataset name*	*atributes*	*rating field*	*item_field*	*main atribute*	*relevant_item*
Tipo	*string*	*array*	*string*	*string*	*string*	*object*

Tabla 5-3: Tabla files.

5.3.4 Tabla *caché*

En esta tabla se almacenarán las métricas una vez calculadas por primera vez.

Clave	*dataset name*	*run name*	*filter field*	*filter condition*	*filter value*	*metric*	*cutoff*	*value*
Tipo	*string*	*string*	*string*	*string*	*string*	*string*	*number*	*number*

Tabla 5-4: Tabla cache.

Los ficheros se guardan en disco. Dado que puede haber ficheros con el mismo nombre, se cambia el nombre por una combinación del id del dataset más el nombre. En el caso de las ejecuciones, el id de la ejecución más su nombre.

6 Integración, pruebas y resultados

6.1 Pruebas unitarias

Las pruebas unitarias son una forma de comprobar que un fragmento de código concreto tiene el funcionamiento esperado. Hay dos tipos de pruebas unitarias: caja negra, en la cual se espera una salida válida, y caja blanca, en la cual se verifica el correcto funcionamiento de todas las secuencias. Se ha hecho uso de las pruebas de caja negra.

Para la librería de evaluación como para la librería de manejo de matrices, se crearon pruebas unitarias. Se han usado las librerías mocha [23] y chai [17] para la realización de tests y la librería nyc [32] para saber la cobertura del código. Se ha medido la cobertura de código como unidad para ver la calidad de los tests.

6.1.1 Librería de evaluación

Se comparó los resultados de las métricas con la librería RiVal [31]. Se ha considerado que los resultados pueden tener un 5% de error máximo respecto a RiVal, ya que la forma de ordenación de las recomendaciones al calcular las métricas es distinta: RiVal, en caso de empate en la valoración de las recomendaciones, las ordena según el ítem. Por el contrario, esta librería no ordena en caso de empate.

Ejecutando los tests con la cobertura de código, salió un resultado de un 97% de cobertura.

```
describe('Metrics tests', function() {
    const error = 0.05;
    const runs = [
        {it: 'MAE', expected: 0.87, value: recsys.mae()},
        {it: 'RMSE', expected: 1.11, value: recsys.rmse()},
        {it: 'P@10', expected: 0.03, value: recsys.precision_at_k(10)},
        {it: 'R@10', expected: 0.01, value: recsys.recall_at_k(10)},
        {it: 'F@10', expected: 0.03, value: recsys.f_at_k(10)},
        {it: 'MRR@10', expected: 0.06, value: recsys.mrr(10)},
        {it: 'MAP@10', expected: 0.00, value: recsys.map(10)},
        {it: 'NDCG@10', expected: 0.02, value: recsys.ndcg(10)},
    ];

    runs.forEach(function (run) {
        it(run.it, function () {
            expect(run.value).to.be.lte(run.expected + error);
            expect(run.value).to.be.gte(run.expected - error);
        });
    });
});
```

Figura 6-1: Tests sobre las métricas

6.1.2 Librería manejo matrices

Ejecutando los tests con la cobertura de código, salió un resultado de un 91% de cobertura.

```
it('limit test', function () {
    let expected_data = [];

    let expected_row = [];
    expected_row['field1'] = 1;
    expected_row['field2'] = 2;
    expected_row['field3'] = 2;

    expected_data.push(expected_row);
    expect(matrix.limit(1).get()).to.deep.equal(expected_data)
});
```

Figura 6-2: Ejemplo de un test de la función *limit*

6.2 Pruebas de integración

Una vez realizados los tests unitarios, se realizaron las pruebas de integración. Estas pruebas sirven para verificar el correcto funcionamiento de varios componentes que actúan en conjunto.

La mayoría de estas pruebas se han realizado contra el API con el programa Postman [27]. Los endpoints en los que debería subirse un archivo, se han realizado con el navegador, por la simplicidad de poder enviarlo desde este y la dificultad de enviarlo con Postman. En ellas se ha comprobado el correcto funcionamiento de cada uno de los endpoints desarrollados.

1. Endpoint */api/home*

    - Acción: GET
    - Prueba: Contrastar que se devuelve el número de datasets y de ejecuciones que hay en la base de datos.
    - Prueba: Contrastar que se devuelve correctamente el número de métricas consultadas.

2. Endpoint */api/datasets*

    - Acción: GET
    - Prueba: Contrastar que se devuelven los datasets que hay en la base de datos.

3. Endpoint */api/datasets/:name*

    - Acción: GET
    - Prueba: Contrastar que se devuelve el dataset que se ha pedido en el parámetro name.
    - Prueba: Contrastar que se devuelve correctamente el número de métricas consultadas sobre el dataset pedido.

4. Endpoint */api/datasets*

    - Acción: POST
    - Prueba: Contrastar que se introduce el dataset en la tabla datasets y las características de los archivos en la tabla files.

5. Endpoint */api/datasets/:name*

    - Acción: DELETE
    - Prueba: Contrastar que se elimina el dataset que se ha pedido en el parámetro name.

6. Endpoint */api/datasets/:dataset_name/runs*

    - Acción: GET

- Prueba: Contrastar que se devuelven las ejecuciones que hay del dataset pedido por parámetro dataset_name en la base de datos.

7. Endpoint */api/datasets/:dataset_name/runs/:name*

    - Acción: GET
    - Prueba: Contrastar que se devuelve la ejecución que se ha pedido en el parámetro name.

8. Endpoint */api/datasets/:dataset_name/runs*

    - Acción: POST
    - Prueba: Contrastar que se introduce la ejecución en la tabla runs.

9. Endpoint */api/datasets/:dataset_name/runs/:name*

    - Acción: DELETE
    - Prueba: Contrastar que se elimina la ejecución que se ha pedido en el parámetro name.

10. Endpoint /api/datasets/:dataset_name/runs/:name/*metrics*

    - Acción: GET
    - Prueba: Contrastar que se calcula la métrica pedida por parámetro con el cutoff pedido por parámetro.
    - Prueba: Contrastar que las métricas se introducen en la tabla caches cuando se calculan por primera vez.
    - Prueba: Contrastar que la segunda vez que se pide una métrica, se saque de la tabla caches y no se calcule.
    - Prueba: Contrastar que las métricas se ejecutan y son correctas aplicándoles un filtro.

11. Endpoint */api/datasets/:dataset_name/compare_runs*

    - Acción: GET
    - Prueba: Contrastar que se calcula la métrica pedida por parámetro con el cutoff pedido por parámetro.
    - Prueba: Contrastar que las métricas se introducen en la tabla caches cuando se calculan por primera vez.
    - Prueba: Contrastar que la segunda vez que se pide una métrica, se extrae de la tabla caches y no se calcule.

Una vez ejecutadas estas pruebas, se corrigieron los errores que se detectaron, y se volvieron a ejecutar.

6.3 Pruebas de sistema

Por último, se probó la aplicación de forma general, comprobando que todos los requisitos funcionales se cumplen.

RF1 – Almacenamiento de datasets.

- Prueba: Se comprueba que se puede almacenar un dataset desde el formulario de subida.
- Prueba: Se comprueba que, al haber un campo incorrecto, se marca como erróneo.
- Prueba: Se comprueba que, si un campo estaba marcado como erróneo y se vuelve a introducir correctamente, se desmarca como erróneo.
- Prueba: Se comprueba que, una vez reciba la respuesta del servidor, se redirige hacia la sección donde se muestran todos los datasets.

RF2 – Consulta de datasets.

- Prueba: Se comprueba que se puede acceder a la visualización de un dataset.
- Prueba: Se comprueba que se pueden ver los atributos de un dataset.
- Prueba: Se comprueba que se puede ver el número de veces que se han ejecutado las distintas métricas sobre un dataset.

RF3 – Búsqueda de datasets.

- Prueba: Se comprueba que se puede realizar la búsqueda de datasets.
- Prueba: Se comprueba que los datasets encontrados se corresponden con la búsqueda introducida.

RF4 – Borrado de datasets.

- Prueba: Se comprueba que se puede eliminar un dataset.

RF5 – Subida de ejecuciones con recomendaciones.

- Prueba: Se comprueba que se puede almacenar una ejecución desde el formulario de subida.
- Prueba: Se comprueba que, al haber un campo incorrecto, se marca como erróneo.
- Prueba: Se comprueba que, si un campo estaba marcado como erróneo y se vuelve a introducir correctamente, se desmarca como erróneo.
- Prueba: Se comprueba que, una vez reciba la respuesta del servidor, se redirige hacia la sección donde se muestran todas las ejecuciones del dataset asociado.

RF6 – Borrado de ejecuciones.

- Prueba: Se comprueba que se puede eliminar una ejecución.

RF7 – Consulta de ejecuciones

- Prueba: Se comprueba que se pueden seleccionar las métricas que se quieren consultar.
- Prueba: Se comprueba que se puede aplicar cutoffs a las métricas seleccionadas.
- Prueba: Se comprueba que se puede filtrar los datos por uno de los atributos.
- Prueba: Se comprueba que se ven los gráficos descritos.
- Prueba: Se comprueba que se puede borrar una fila de la tabla.
- Prueba: Se comprueba que se puede borrar la tabla.

RF8 – Comparación de ejecuciones

- Prueba: Se comprueba que se pueden seleccionar las métricas que se quieren consultar.
- Prueba: Se comprueba que se puede aplicar cutoffs a las métricas seleccionadas.
- Prueba: Se comprueba que se ven los gráficos descritos.

RF9 – Creación de una página principal.

- Prueba: Se comprueba que se visualizan los últimos cuatro datasets, ordenados por fecha de subida.
- Prueba: Se comprueba que se visualiza el número total de dataset y el número total de ejecuciones.
- Prueba: Se comprueba que se puede ver el número de veces que se han ejecutado las distintas métricas.

RF10 – Creación de una página que contendrá todos los datasets almacenados, ordenados por fecha de creación.

- Prueba: Se comprueba que se pueden ver todos los datasets almacenados.

RF11 – Creación de una página que contendrá todas las ejecuciones almacenadas, ordenadas por fecha de creación.

- Prueba: Se comprueba que se pueden ver todas las ejecuciones sobre un dataset almacenadas.

RN12 – Información sobre las métricas: Se mostrará una pequeña descripción sobre las métricas, la cual contendrá el objetivo de esta, y su fórmula.

- Prueba: Se comprueba que se puede ver la descripción de las métricas y sus fórmulas.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

Este trabajo se desarrolló para dar una herramienta funcional a desarrolladores de sistemas de recomendación, con el objetivo de facilitar su trabajo, sobre todo la parte derivada de la evaluación de los mismos. Para llegar a este objetivo, todo lo relacionado con los sistemas de recomendación, así como el desarrollo de la aplicación con JavaScript, Node.js y React.js ha sido aprendido desde cero, por lo que este trabajo ha sido una gran motivación para el aprendizaje de estos conceptos.

Se ha realizado un estudio de los sistemas de recomendación, poniendo énfasis en su evaluación. Se ha aprendido sobre los tipos de sistemas, los algoritmos que usan estos sistemas para generar recomendaciones y, por supuesto, sobre su evaluación.

Por otro lado, se ha desarrollado todo el proyecto en JavaScript, un lenguaje que el autor no había utilizado nunca de forma exhaustiva, así como tampoco conocía las librerías para desarrollar la vista de una aplicación ni el entorno de Node.js para el desarrollo de la parte del servidor.

A partir de esto, se han conocido librerías importantes para la evaluación, como RiVal, librerías para el desarrollo, como React.js, Mongoose.js o Napa.js, y librerías para la realización de pruebas, como Mocha.js y Chai.js.

En resumen, este trabajo ha sido una importante fuente de aprendizaje para el autor.

7.2 Trabajo futuro

Se pueden realizar diversas mejoras sobre la aplicación:

- La principal mejora sería la creación de un sistema completo de recomendaciones. Este sistema permitiría la creación y exportación de recomendaciones a partir de un dataset. Desarrollando esto, se podría prescindir de aplicaciones externas.

- Subida de evaluaciones desde un fichero externo. Dado que un desarrollador puede haber evaluado sus sistemas de recomendación desde otra herramienta, permitir que pueda subir esas evaluaciones y poder usar las ventajas que aporta esta aplicación.

- Agregación de usuarios. Actualmente sólo se permite el uso de la aplicación por parte de un usuario. Sería interesante la implementación de un sistema de usuarios y que cada uno pudiera usarla al mismo tiempo, tener sus propios datasets y sus propias ejecuciones.

Referencias

[1] Adamopoulos, Panagiotis and Tuzhilin, Alexander, "On Unexpectedness in Recommender Systems: Or How to Expect the Unexpected", 5 th ACM International Conference on Recommender Systems, 2011, Chicago, USA.

[2] Amatriain, Xavier, Jaimes, Alejandro, Oliver, Nuria, and Pujol, Josep M.. "Data Mining Methods for Recommender Systems" Recommender Systems Handbook, 2011.

[3] Bellogín, Alejandro, "Recommender systems", Performance prediction and evaluation in Recommender Systems: an Information Retrieval perspective, Universidad Autónoma de Madrid, Madrid, España, 2012.

[5] Bellogín, Alejandro, "Evaluation of recommender systems", Performance prediction and evaluation in Recommender Systems: an Information Retrieval perspective, Universidad Autónoma de Madrid, Madrid, España, 2012.

[5] Bellogín, Alejandro, Castells, Pablo and Cantador, Iván, "Precision-Oriented Evaluation of Recommender Systems: An Algorithmic Comparison", RecSys'11, Chicago, Illinois, USA, 2011.

[6] Cantador, Iván, Bellogín, Alejandro, and Vallet, David, "Content-based Recommendation in Social Tagging Systems", RecSys '10: Proceedings of the fourth ACM conference on Recommender systems, ACM, 2010.

[7] Hernández del Olmo, Félix and Gaudioso, Elena, "Evaluation of recommender systems: A new approach", Elsevier Ltd, 2007.

[8] Herlocker, Jonathan L., Konstan, Joseph A., Terveen, Loren G. and Riedl, John T., "Evaluating Collaborative Filtering Recommender Systems", ACM Transactions on Information Systems, Vol. 22, No. 1, 2004.

[9] Hripcsak, George, and Rothschild, Adam S. "Agreement, the F-Measure, and Reliability in Information Retrieval", Journal of the American Medical Informatics Association Volume 12, Number 3, Jun 2005.

[10] Koren, Yehuda and Bell, Robert "Advances in Collaborative Filtering", Recommender Systems Handbook, 2011.

[11] Linden, Greg, Smith, Brent, and York, Jeremy "Amazon.com Recommendations, Item-to-Item Collaborative Filtering" IEEE Computer Society, 2003.

[12] Lops, Pasquale, de Gemmis, Marco and Semeraro, Giovanni, "Content-based recommender Systems: State of the Art and Trends", Recommender Systems Handbook, 2011.

[13] Piao, Guangyuan, and Breslin, John G., “Measuring Semantic Distance for Linked Open Data-enabled Recommender Systems”, 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, 2016.

[14] Sarwar, Badrul M., Karypis, George, Konstan, Joseph, and Riedl, John “Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering”, University of Minnesota, Minneapolis, USA.

[15] Velupillai, Sumithra “Annotation and Classification of Swedish Medical Records”, Stockholm University, Stockholm, Sweden, 2012.

[16] Bulma, https://bulma.io/

[17] Chai.js http://www.chaijs.com/

[18] “Container Pattern”, best-practice-software-engineering.ifs.tuwien.ac.at, 2013, http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/container.html

[19] Google Charts, https://developers.google.com/chart/

[20] GroupLens Research, https://grouplens.org/datasets/movielens/

[21] Material Design, https://material.io/

[22] Matrix.js, https://github.com/ricardomoratomateos/matrix

[23] Mocha.js https://mochajs.org/

[24] Mongoose.js, http://mongoosejs.com/

[25] Napa.js, https://github.com/Microsoft/napajs

[26] Nyc, https://github.com/istanbuljs/nyc

[27] Postman, https://www.getpostman.com/

[28] React.js, https://reactjs.org/

[29] React Google Charts, https://github.com/RakanNimer/react-google-charts

[30] RecSys.js, https://github.com/ricardomoratomateos/lib

[31] RiVal, https://github.com/recommenders/rival

[32] TechMagic Development Studio, “React vs Angular5 vs Vue.js – What to choose in 2018?”, medium.com, 2018, https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

[32] Webpack, https://webpack.js.org/

Glosario

API	Application Programming Interface
Backend	Parte de servidor de una aplicación web
Cutoff	Corte a aplicar sobre los datos al evaluar una métrica.
Dataset	Conjunto de datos usado como entrada en un sistema de recomendación.
Ejecución	Conjunto de datos generados al ejecutar un sistema de recomendación.
Endpoint	Punto de acceso al API
Frontend	Parte visual de una aplicación web

Anexos

A Manual de instalación

Instalación de la aplicación en un entorno sobre Windows:

- Paso 1, Instalación de Node.js: Descargar el instalador de la página web oficial[1] y ejecutarlo.
- Paso 2, Instalación de MongoDB: Descargar el instalador de la página web oficial[2] y ejecutarlo.
- Paso 3, Descargar la aplicación del repositorio de Github[3].
- Paso 4, Instalar las dependencias: Con el gestor de paquetes *npm*, ejecutar *npm install* sobre el directorio raíz de la aplicación.
- Paso 5, Ejecutar MongoDB en local.
- Paso 6, Ejecutar la aplicación: Con el comando *node server.js* sobre el directorio *src* de la aplicación.
- Paso 7, Acceder a la aplicación desde el navegador, desde la url http://localhost:3000/app

[1] https://nodejs.org/es/download/

[2] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

[3] https://github.com/ricardomoratomateos/app

B Imágenes de la aplicación

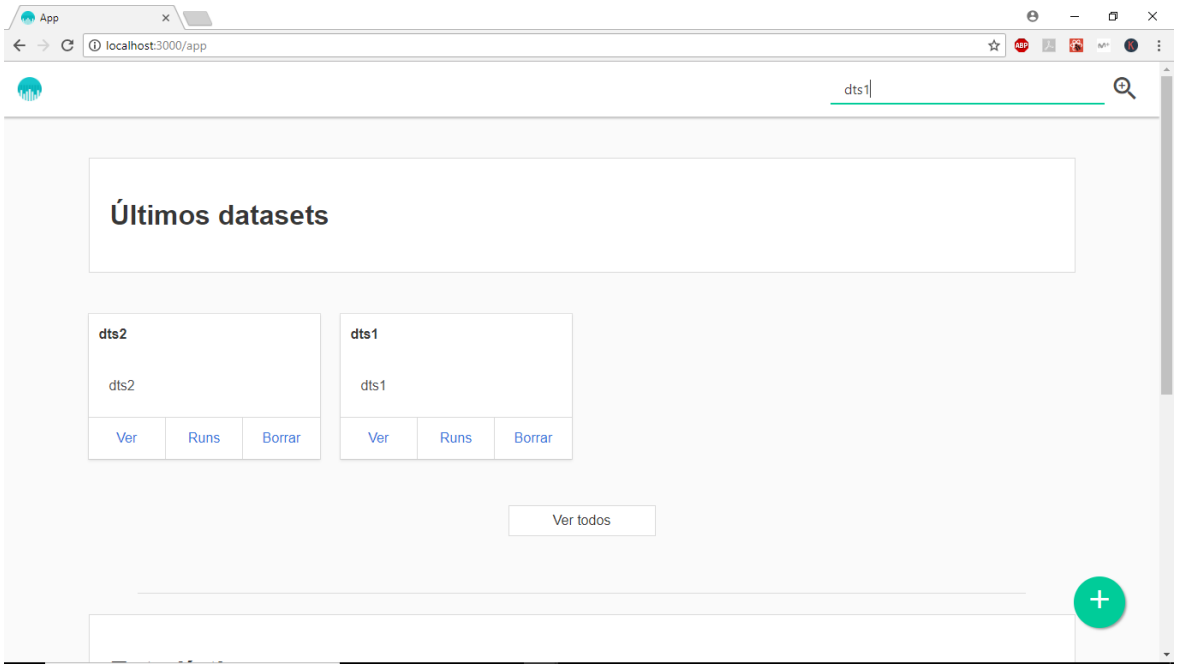

Figura B-1: Captura – Pantalla principal

La pantalla principal de la aplicación se muestra en la Figura B-1. Está relacionada con los requisitos funcionales RF9 y RF9.1, donde se muestra un listado con los últimos datasets subidos. Pulsando el botón de borrar se eliminan de la aplicación, como indica el requisito funcional RF2. Pulsando el botón *Runs*, podremos ver el listado de ejecuciones de ese dataset, como indica el requisito funcional RF11.

Figura B-2: Captura – Pantalla principal, estadísticas

La Figura B-2 muestra la pantalla principal de la aplicación, relacionada con los requisitos funcionales RF9, RF9.2 y RF9.3, mostrando el número total de datasets, el número total de ejecuciones y el número de veces que se ha ejecutado cada métrica.

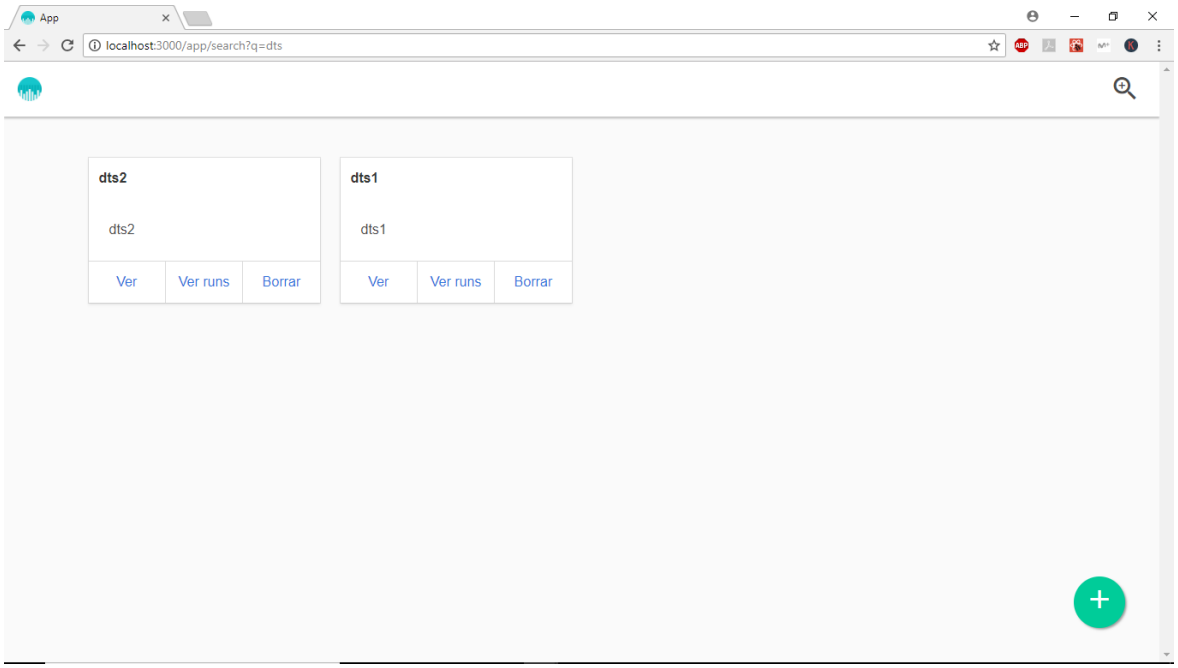

Figura B-3: Captura – Búsqueda de datasets

Al acceder a la pantalla de búsqueda de datasets, relacionada con el requisito funcional RF3, se podrán ver aquellos datasets que cumplan con la consulta introducida (ver Figura B-3). Pulsando el botón *borrar* se eliminan de la aplicación, como indica el requisito funcional RF2.

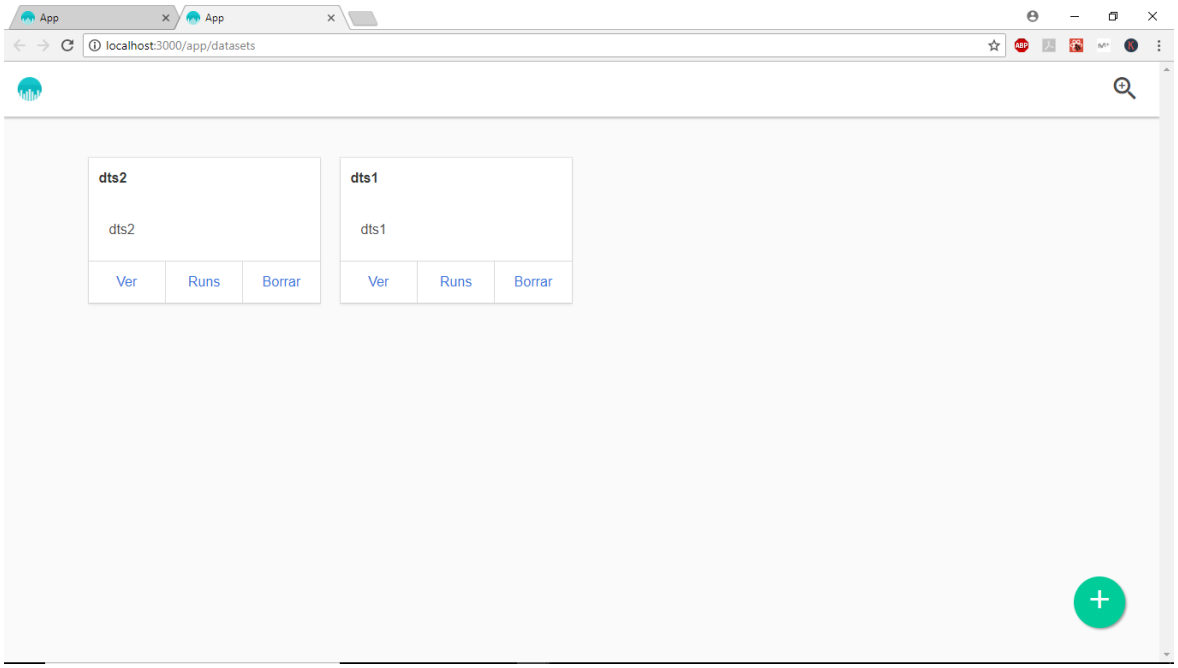

Figura B-4: Captura – Listado de datasets

En esta figura, B-4, se puede ver el listado de todos los datasets almacenados en la aplicación, relacionada con el requisito funcional RF10. Aunque parezca igual que la anterior, es mera coincidencia. Se puede apreciar en la url que son listados distintos. Pulsando el botón de *borrar* se eliminan de la aplicación, como indica el requisito funcional RF2.

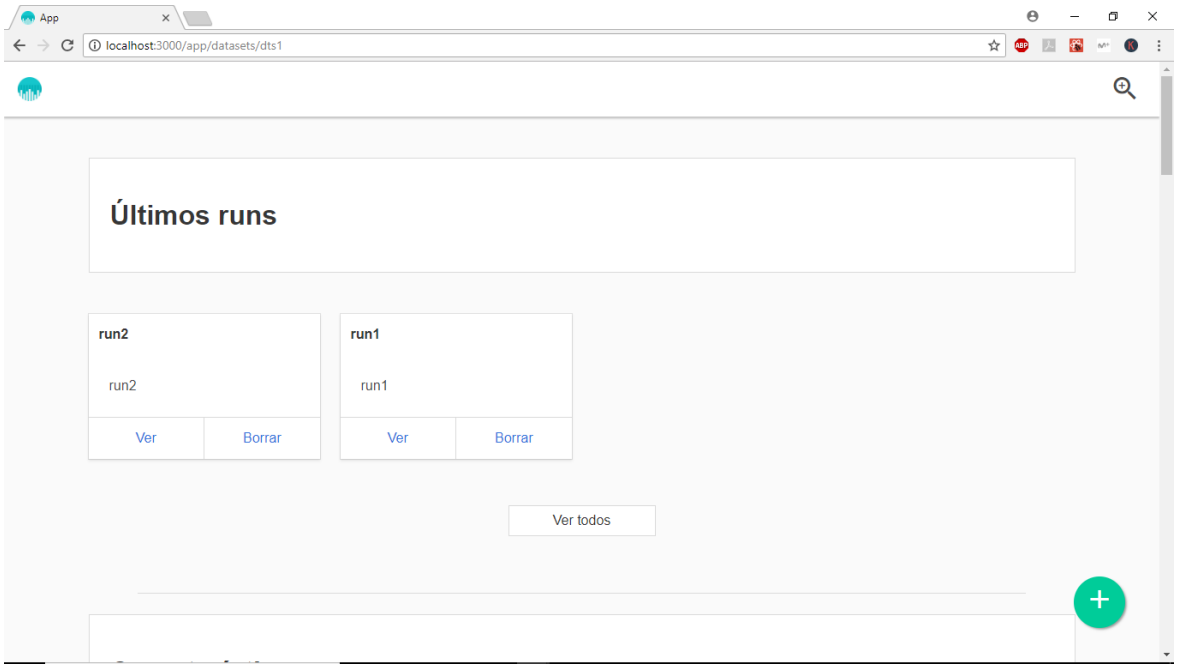

Figura B-5: Captura – Vista de un dataset, últimas ejecuciones subidas

En la figura B-5 se puede ver el listado de las últimas ejecuciones almacenadas de un dataset en la aplicación, relacionada con los requisitos funcionales RF2 y RF2.1.

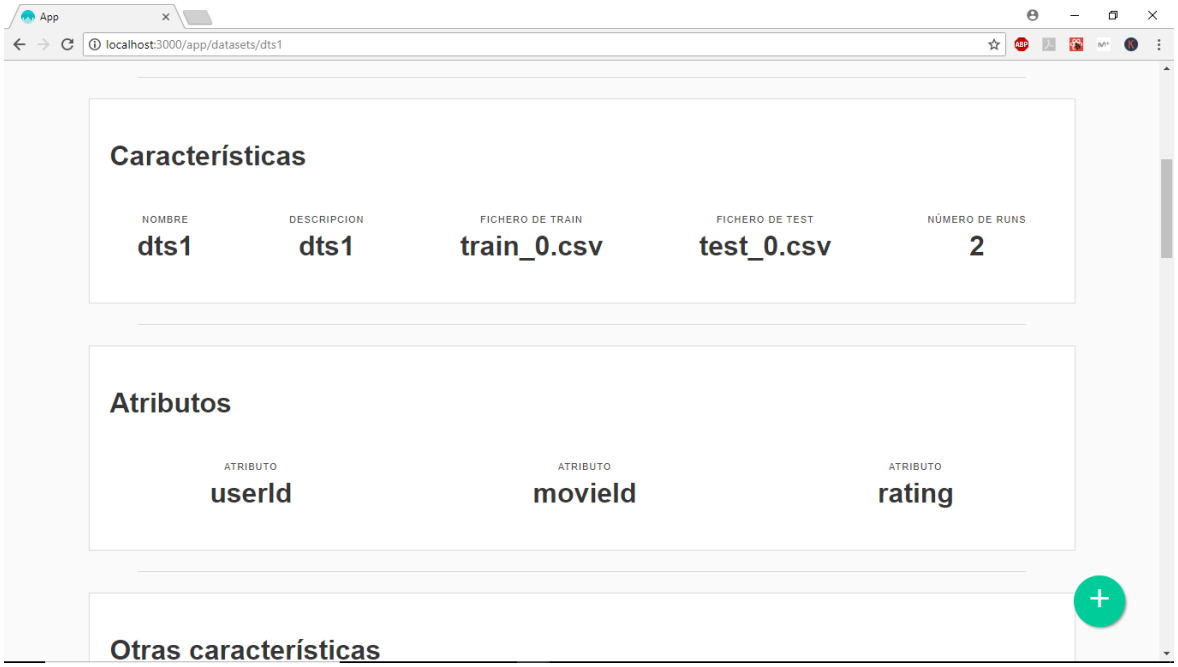

Figura B-6: Captura – Vista de un dataset, características y atributos

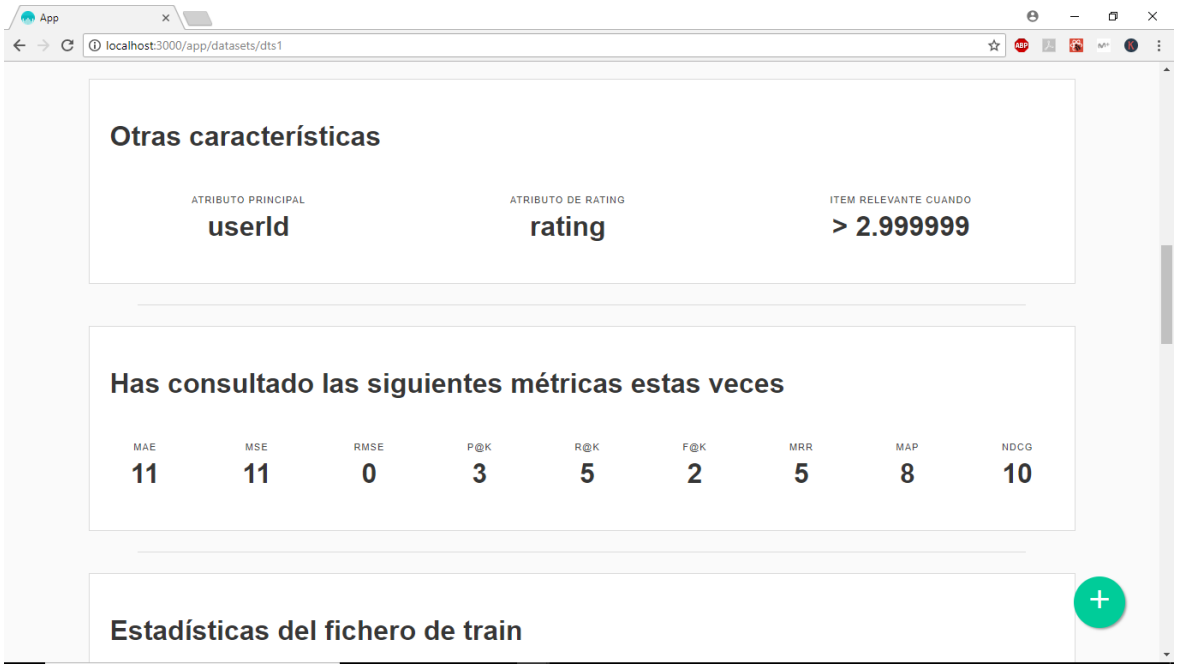

Figura B-7: Captura – Vista de un dataset, otras características y consultas de métricas

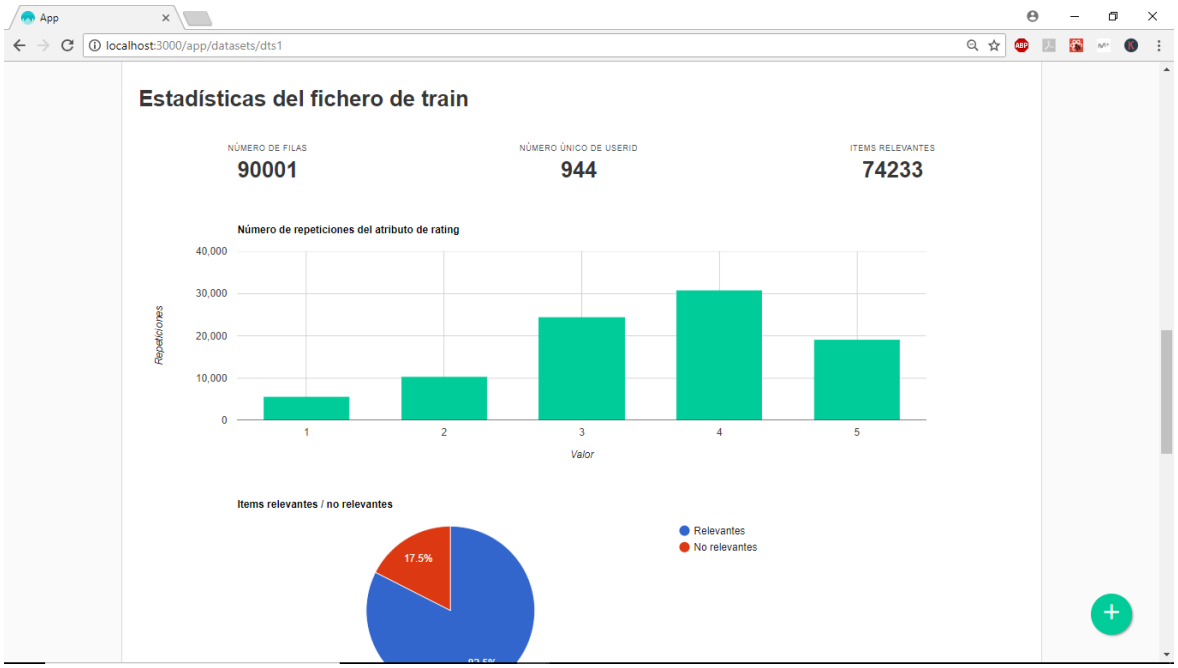

Figura B-8: Captura – Vista de un dataset, estadísticas del fichero de entrenamiento

Al consultar un dataset se pueden ver los atributos de un dataset (nombre, descripción, atributos, cuándo es relevante, etc.), relacionada con los requisitos funcionales RF1.1, RF2 y RF2.2 (ver Figura B-6 y Figura B-7) y el número de veces que se ha ejecutado cada métrica, relacionado con el requisito funcional RF2 y RF2.3 (ver Figura B-7).

También se muestran las características del fichero de entrenamiento (ver Figura B-8), como el número de entradas, el número de ítems relevantes o el número de repeticiones de los valores del atributo de rating, características descritas en el requisito funcional RF2.4. Hay otra pantalla similar para el fichero de test, descrita en el requisito funcional RF2.5.

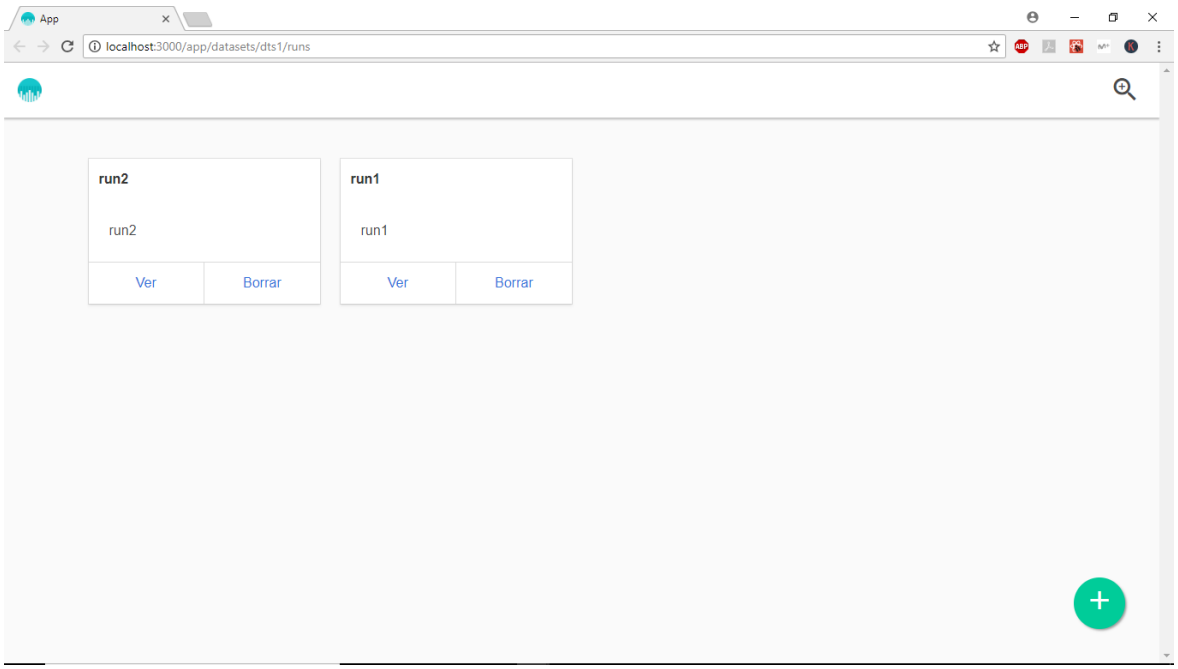

Figura B-9: Captura – Listado de ejecuciones de un dataset

En la Figura B-9 se puede ver el listado de ejecuciones de un dataset, descrito en el requisito funcional RF11. Pulsando el botón *borrar* se eliminan de la aplicación, como indica el requisito funcional RF6.

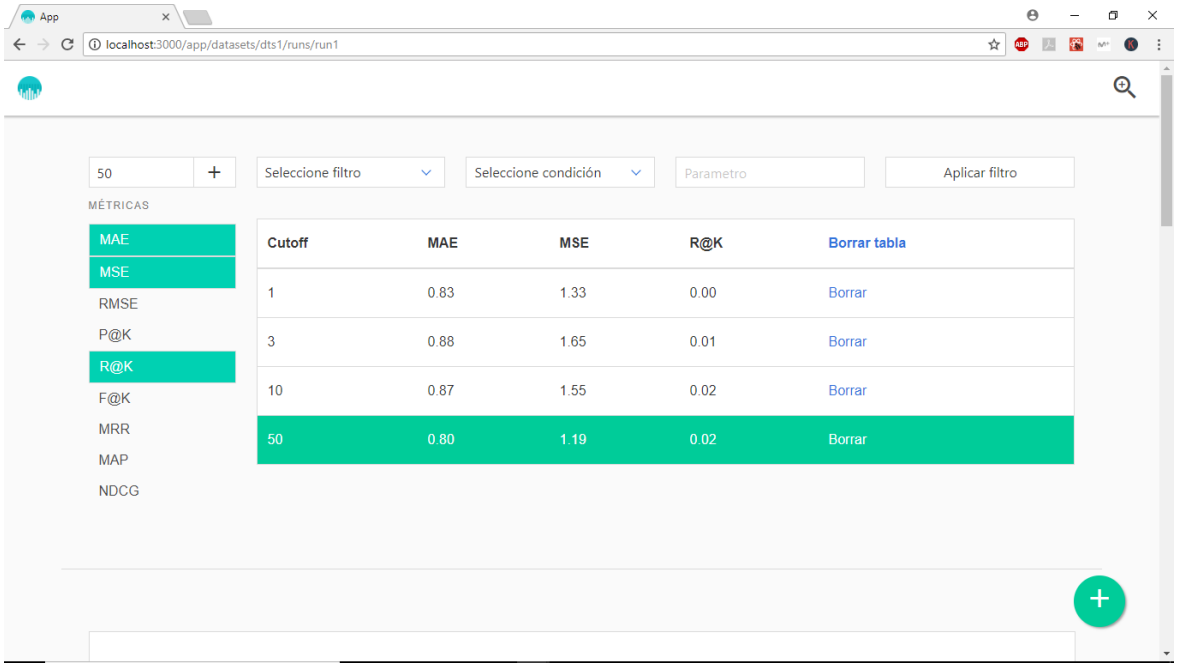

Figura B-10: Captura - Vista de una ejecución, comparativa de distintas métricas y cutoffs

Figura B-11: Captura - Vista de una ejecución, comparativa de una métrica con distintos cutoffs

Respecto a la consulta de ejecuciones, descrito en el requisito funcional RF7, se puede ver en la Figura B-10 la consulta sobre las métricas elegidas en el menú de la izquierda, como

indica el requisito funciona RF7.1. Se puede aplicar uno o varios cutoffs, como indica el requisito funcional RF7.2, escribiendo el cutoff en el recuadro de la izquierda y pulsando sobre el botón de más que está a su lado. Se puede aplicar filtros, como indica el requisito funcional RF7.3. Se puede borrar la tabla o borrar sus filas.

En la Figura B-11 se muestra una comparativa en forma de gráfico de una métrica, en concreto la métrica MAE, para varios cutoffs, como indica el requisito funcional RF7.4. Por cada métrica elegida, habrá un gráfico comparativo. También, se puede apreciar una pequeña descripción de la métrica, como indica el requisito funcional RF12.

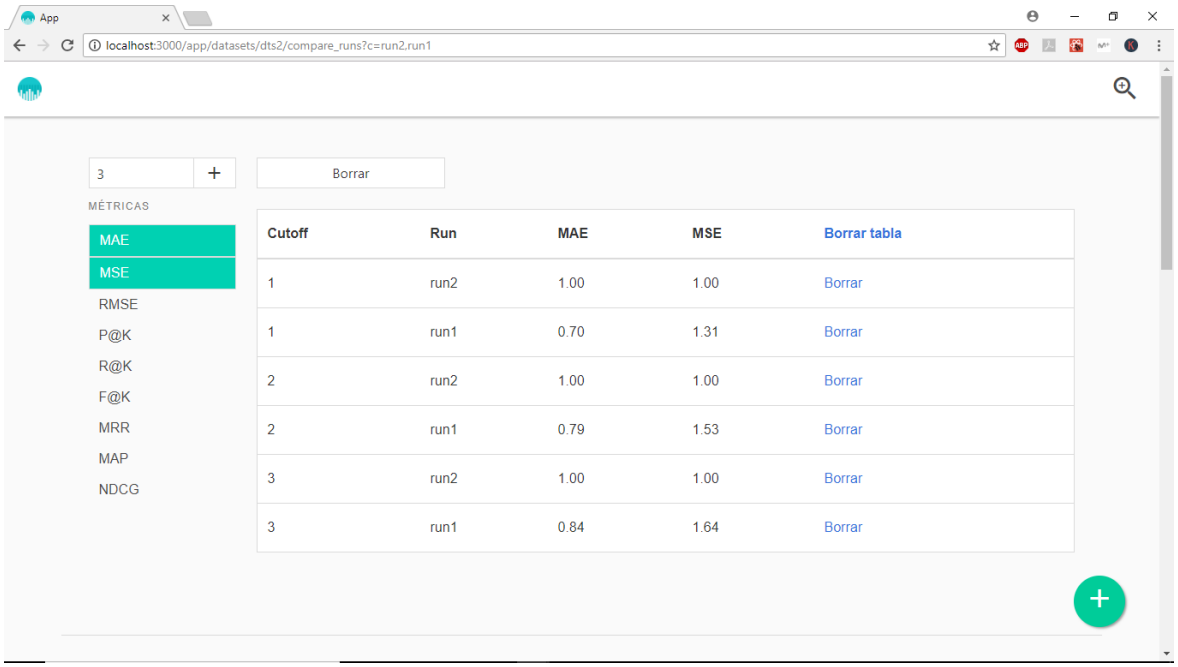

Figura B-12: Captura – Vista de una comparación de dos ejecuciones de un dataset, comparando distintas métricas

Figura B-13: Captura – Vista de una comparación de dos ejecuciones de un dataset, comparando una métrica entre las distintas ejecuciones

La pantalla de comparación de ejecuciones descrita en el requisito funcional RF8 y visible en las Figuras B-12 y B-13, muestra la comparativa sobre las métricas elegidas en el menú de la izquierda, como indica el requisito funciona RF8.1. Se puede aplicar uno o varios cutoffs, como indica el requisito funcional RF8.2, escribiendo el cutoff en el recuadro de la izquierda y pulsando sobre el botón de *más* que está a su lado. Se puede borrar la tabla o borrar sus filas.
También muestra una comparativa en forma de gráfico de una métrica, en concreto la métrica MAE, para varios cutoffs y las ejecuciones que se han elegido previamente, como indica el requisito funcional R8.3. Por cada métrica elegida, habrá un gráfico comparativo y una pequeña descripción de la métrica, como indica el requisito funcional RF12.

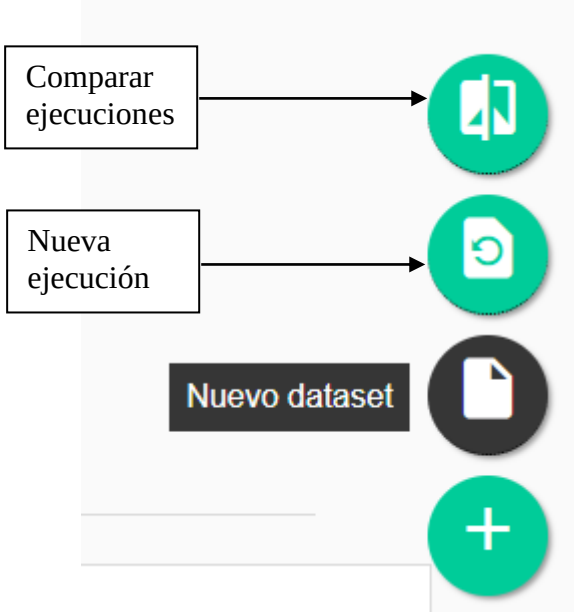

Figura B-14: Captura – Vista extendida del botón principal.

Por último, en la Figura B-14, se muestra un botón, el cual es accesible desde toda la aplicación. Este botón nos da tres opciones: *Nuevo dataset*, en el que aparecerá una ventana modal con un formulario para subir un dataset a la aplicación, (ver requisito funcional RF1), *Nueva ejecución*, en el que aparecerá una ventana modal con un formulario para subir una nueva ejecución asociada a un dataset (ver requisito funcional RF5), y *Comprar ejecuciones*, en el que aparecerá un formulario que mostrará el listado de los datasets que hay en la aplicación y una vez seleccionado uno, mostrará sus ejecuciones, pudiendo seleccionar tantas como queramos y compararlas (ver requisito funcional RF8).