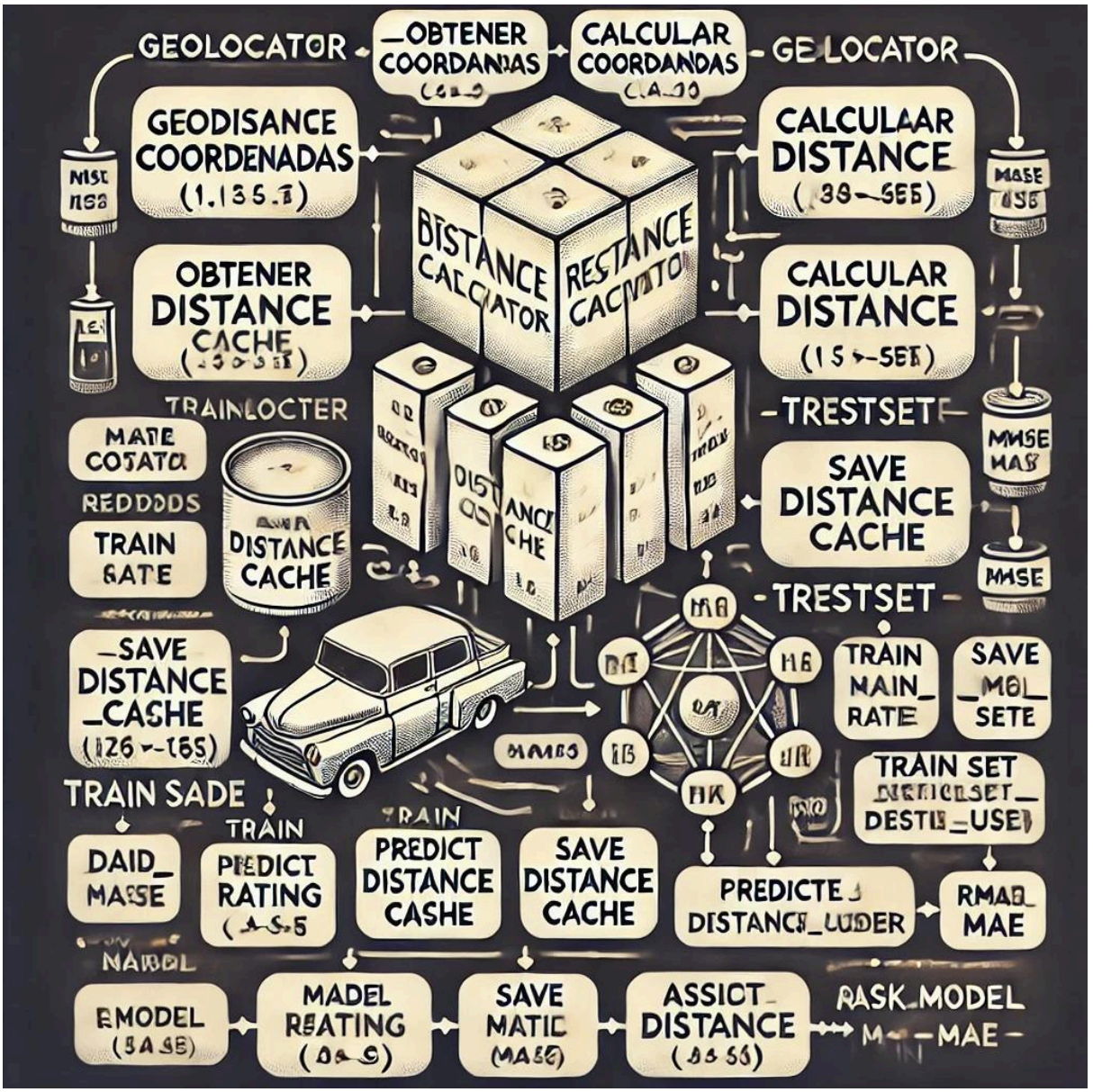
Sistema de Recomendación de Coches



ÍNDICE

Introducción

Problema

Elegir el coche ideal puede ser una tarea abrumadora para los usuarios debido a la amplia variedad de opciones disponibles y la cantidad de factores a considerar. Aspectos como el presupuesto, las características técnicas, la ubicación geográfica, y las preferencias personales deben evaluarse cuidadosamente, lo que complica el proceso de decisión. Esto resulta en búsquedas largas, frustrantes y, en ocasiones, en elecciones que no satisfacen completamente las expectativas del comprador. Por lo tanto, es necesario desarrollar una solución que simplifique este proceso y proporcione recomendaciones personalizadas y precisas.

Objetivo

El objetivo principal es desarrollar un programa que nos ayude a encontrar nuestro coche ideal dentro de casi todos los que se encuentran a la venta en el país. Para ello implementamos un sistema híbrido de recomendación de coches que integra diferentes técnicas de recomendación: el filtrado colaborativo, el filtrado basado en contenido y la geolocalización. Este sistema permitirá a los usuarios recibir sugerencias personalizadas que se ajusten a sus preferencias específicas, como el tipo de combustible, presupuesto, tipo de transmisión, etc. Adicionalmente, se busca proporcionar una herramienta intuitiva y fácil de usar que optimice el proceso de búsqueda y selección de coches, ahorrando tiempo y mejorando la experiencia del usuario, por lo que hemos añadido una interfaz que todo el mundo pueda entender, para así llegar a todos los públicos.

Alcance

El sistema está diseñado para usuarios que desean adquirir un coche y buscan una solución personalizada basada en sus preferencias y necesidades. Incluye la capacidad de:

- Cargar datos: Utilizar datasets preexistentes que contengan información de coches disponibles y valoraciones de usuarios.

- Personalización de recomendaciones: Integrar las preferencias individuales del usuario (por ejemplo, marca, precio, kilometraje, potencia del motor, etc.) y asignar pesos según la importancia de cada criterio.

- Predicción de valoraciones: Utilizar modelos colaborativos para estimar la satisfacción esperada de un usuario con respecto a un modelo de coche específico, incluso si no ha interactuado con él antes.

- Consideración geográfica: Incorporar penalizaciones basadas en la distancia entre el usuario y la ubicación del coche para priorizar opciones más cercanas.

- Mostrar resultados: Proveer una lista de coches en base a un puntaje obtenido a partir de las características del usuario y los pesos, asegurándonos que el resultado contiene vehículos que vayan a satisfacer al cliente..

Diseño

Hemos diseñado una arquitectura híbrida para nuestro sistema de recomendación de coches que combina dos enfoques: un sistema basado en filtrado colaborativo, implementado con la librería Surprise, y un sistema basado en características específicas de los coches.

Para personalizar las recomendaciones, planteamos al usuario preguntas sobre sus preferencias en diversas características de los coches (como marca, presupuesto, potencia, entre otras) y el nivel de importancia que asigna a cada una. El usuario puede evaluar cada característica y su peso en una escala de importancia que va del 1 al 10.

Con las respuestas del usuario, se asigna una puntuación a cada coche según cuánto se ajusta a lo que el cliente busca. Las características más importantes tienen mayor peso en la evaluación, bonificando los coches que mejor cumplen con los criterios indicados. Por el contrario, aquellos coches que no se ajustan a requisitos clave, como superar un presupuesto máximo, reciben una penalización que reduce significativamente su puntuación. De esta forma, se genera una lista ordenada de coches según su afinidad con las preferencias del usuario, facilitando una selección más personalizada.

Una vez obtenida esta lista preliminar basada en contenido, el sistema aplica el filtrado colaborativo para estimar las calificaciones que el usuario podría otorgar a cada modelo. Este enfoque utiliza datos históricos de valoraciones de otros usuarios* y permite predecir la afinidad del cliente con modelos específicos en función de patrones de comportamiento similares.

Además, para ajustar las recomendaciones al contexto del usuario, se aplica una penalización geográfica en función de la distancia entre la ubicación del usuario y la localización de los coches disponibles. Esta penalización asegura que los vehículos cercanos tengan una puntuación más favorable, dado que la proximidad puede ser un factor clave en la toma de decisiones.

Finalmente, el sistema combina las tres puntuaciones obtenidas —la similitud de contenido, la predicción colaborativa y la penalización geográfica— en una puntuación híbrida. Cada componente tiene un peso específico: la similitud por contenido influye en un 40%, la predicción colaborativa en un 30%, y la proximidad geográfica en el 30% restante. Los coches se ordenan de mayor a menor según esta puntuación híbrida, ofreciendo al usuario una lista final priorizada que combina sus preferencias personales, la experiencia de otros usuarios y el contexto geográfico. De esta manera, el sistema logra recomendaciones más completas, precisas y adaptadas a las necesidades del usuario.

Requisitos

Para el funcionamiento del programa era necesario un dataset de coches con sus características que obtuvimos del github de DataMarket que ha sacado los datos de la web coches.net. El dataset no era consistente por lo que tuvimos que pre procesarlo:

- Eliminamos los valores nulos.
- Imputamos valores numéricos vacíos mediante Random Forest Regressor.
- Cambiamos el formato de otros como los colores, que dependiendo de la instancia podía estar en formatos distintos.
- Eliminamos columnas innecesarias.

Era necesario un dataset con valoraciones de coches dada por otros usuarios para la librería *Surprise*. Debido a la dificultad de encontrarlo, lo hemos tenido que generar nosotros. En el apartado de **Problemas Encontrados** detallamos en mayor profundidad cómo lo hemos creado.

Para calcular las distancias, empleamos la librería *geopy*, que utiliza la API de Google. Sin embargo, este enfoque incrementa significativamente los tiempos de ejecución. Para optimizar el proceso, implementamos un dataset local que almacena las consultas previamente realizadas. De esta forma, si se introduce una ciudad que ya se había utilizado, no es necesario realizar una nueva búsqueda en Google.

Diagramas UML

Diagrama de clases

En este diagrama se presentan las clases que vamos a emplear junto con sus atributos y métodos. En este caso es un diagrama simple ya que el problema a resolver no necesita una gran complejidad a nivel de clases y no empleamos herencia.

CarRecommenderApp

Interfaz principal de la aplicación que integra todas las funcionalidades para interactuar con el usuario.

HybridRecommender

Combina las puntuaciones del modelo colaborativo, el análisis de contenido y las penalizaciones geográficas para recomendar coches.

CollaborativeFilter

Implementa el filtrado colaborativo utilizando un modelo de aprendizaje basado en el comportamiento de los usuarios.

ContentFilter

Proporciona funcionalidades para calcular la similitud entre las características de los coches y las preferencias del usuario.

GeoUtils

Calcula distancias geográficas entre el usuario y las ubicaciones de los coches, y aplica penalizaciones basadas en la distancia.

DataLoader

Proporciona funciones para cargar datos desde archivos CSV.

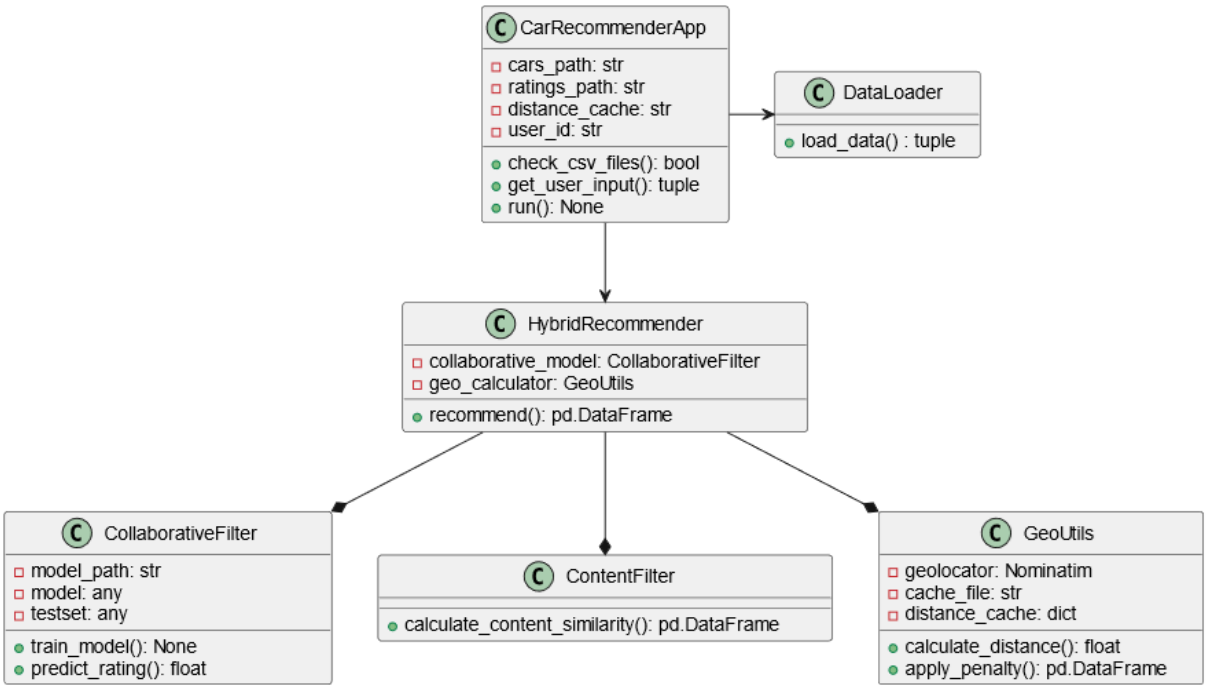


Diagrama de clases

Diagrama de flujo

El diagrama de flujo es el que muestra los pasos que sigue el programa desde que se inicia hasta que da un resultado. Lo hemos dividido en dos partes, una que explica el funcionamiento del programa sin profundizar en la lógica, y otra que explica exclusivamente como funciona el entrenamiento del modelo.

General

El sistema comienza verificando la disponibilidad de tres archivos CSV fundamentales: uno con información de vehículos, otro con valoraciones de usuarios y un tercero con datos las distancias. Si los archivos no están disponibles, se detiene el proceso mostrando un mensaje de error; de lo contrario, los datos se cargan en memoria.

A continuación, el sistema valida si existe un modelo de recomendaciones previamente entrenado. Si está disponible, se utiliza directamente; en caso contrario, se entrena uno nuevo con los datos cargados. Seguidamente, el usuario introduce sus preferencias y asigna pesos según la importancia de cada criterio.

Con esta información, el sistema considera la localización del usuario, revisando si la ciudad está en el caché de distancias. Si los datos existen, se cargan; de lo contrario, se calculan las distancias y se actualiza el caché. Finalmente, utilizando todos los datos procesados, el sistema genera una lista de recomendaciones, mostrando al usuario las cinco mejores opciones de automóviles según sus necesidades y contexto.

Diagrama de flujo del modelo

El diagrama de flujo detalla el proceso diseñado para entrenar un modelo de recomendaciones basado en técnicas de filtrado colaborativo utilizando el algoritmo *SVD* (Singular Value Decomposition).

El proceso inicia verificando si existe un modelo previamente entrenado y almacenado en un archivo. Si el modelo está disponible, se carga directamente desde el archivo, lo que permite evitar el costo computacional de un nuevo entrenamiento, finalizando así el proceso.

En caso de que el modelo no exista, el sistema procede a cargar los datos necesarios desde un archivo CSV que contiene valoraciones de usuarios. Posteriormente, se configura un lector de datos utilizando la biblioteca *Surprise* y se convierte el conjunto de datos en un formato compatible con dicha biblioteca. Los datos se dividen en un conjunto de entrenamiento y uno de prueba utilizando la técnica *train_test_split*, que asegura una separación adecuada para evaluar la precisión del modelo.

Seguidamente, se define el modelo *SVD* configurando los parámetros necesarios para optimizar su rendimiento. Con estos parámetros, se entrena el modelo utilizando el conjunto de entrenamiento. Una vez completado el entrenamiento, el modelo se guarda en un archivo para su reutilización en futuras ejecuciones.

Finalmente, el proceso concluye asegurando que un modelo entrenado (ya sea cargado o recién creado) está listo para ser utilizado en el sistema de recomendaciones.

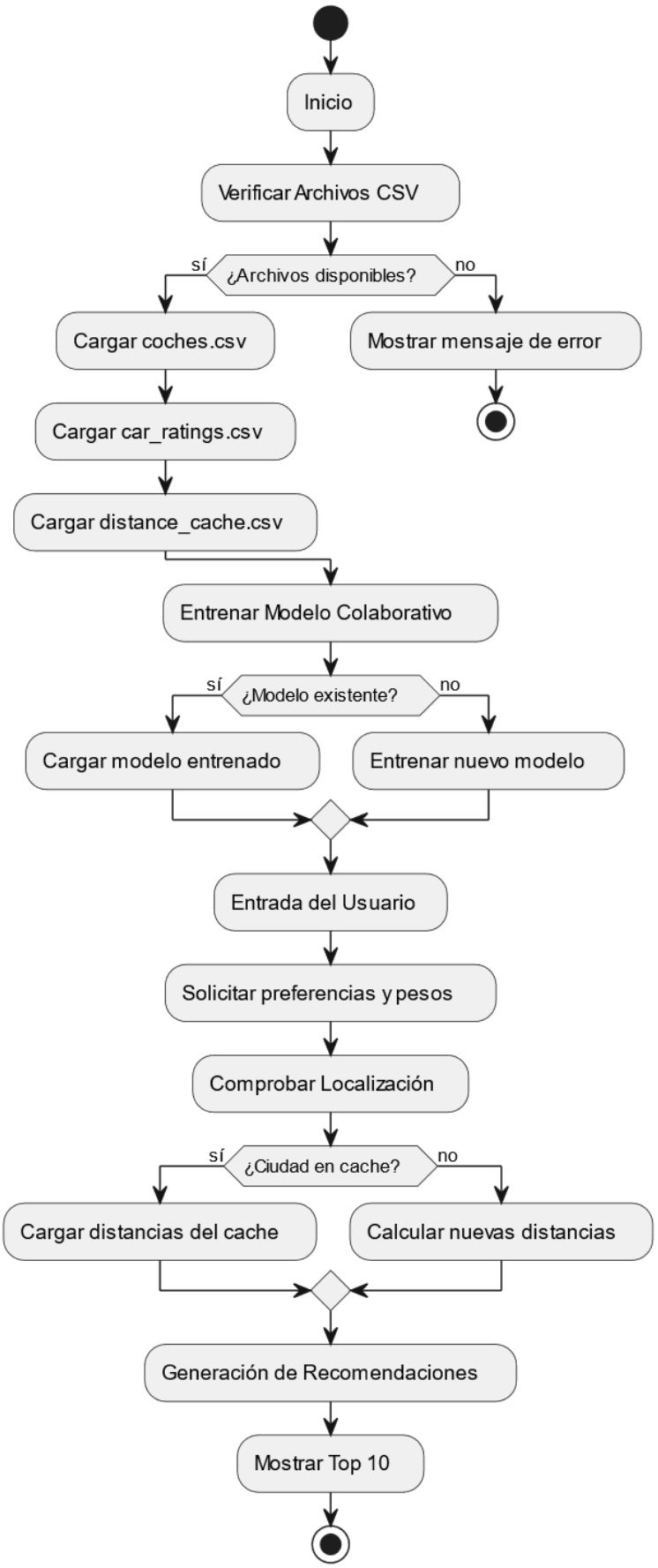


Diagrama de flujo general

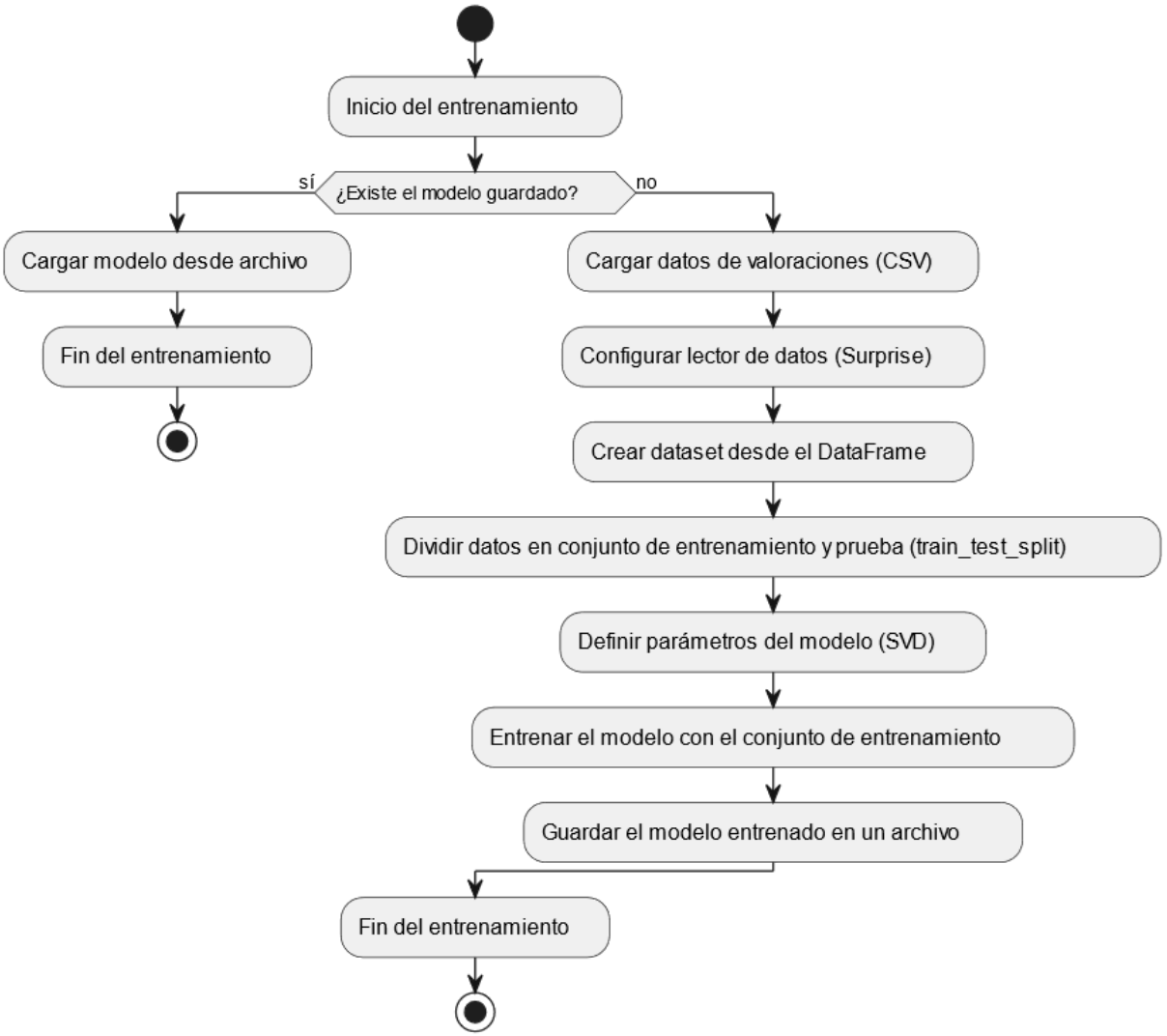


Diagrama de flujo del modelo

Diagrama de casos de uso

Este diagrama muestra cómo un cliente interactúa con el programa.

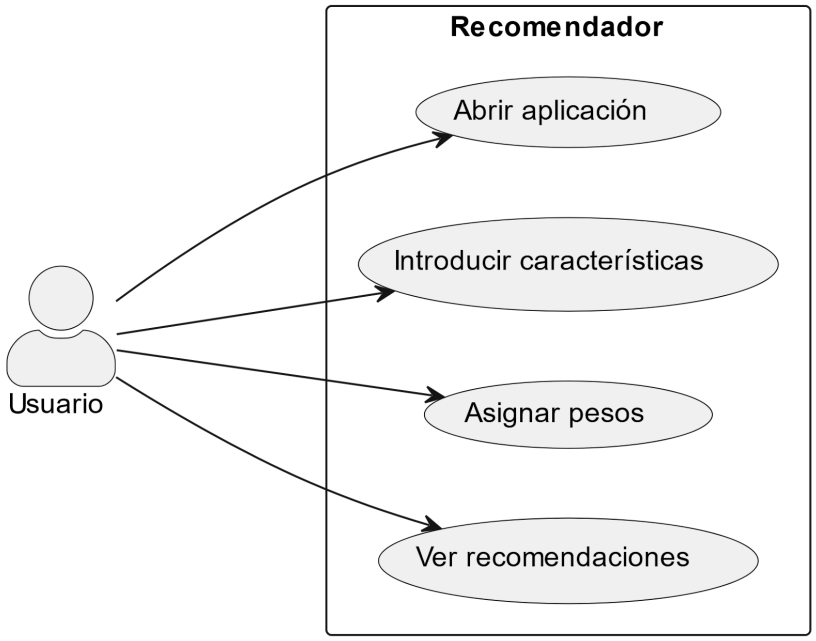
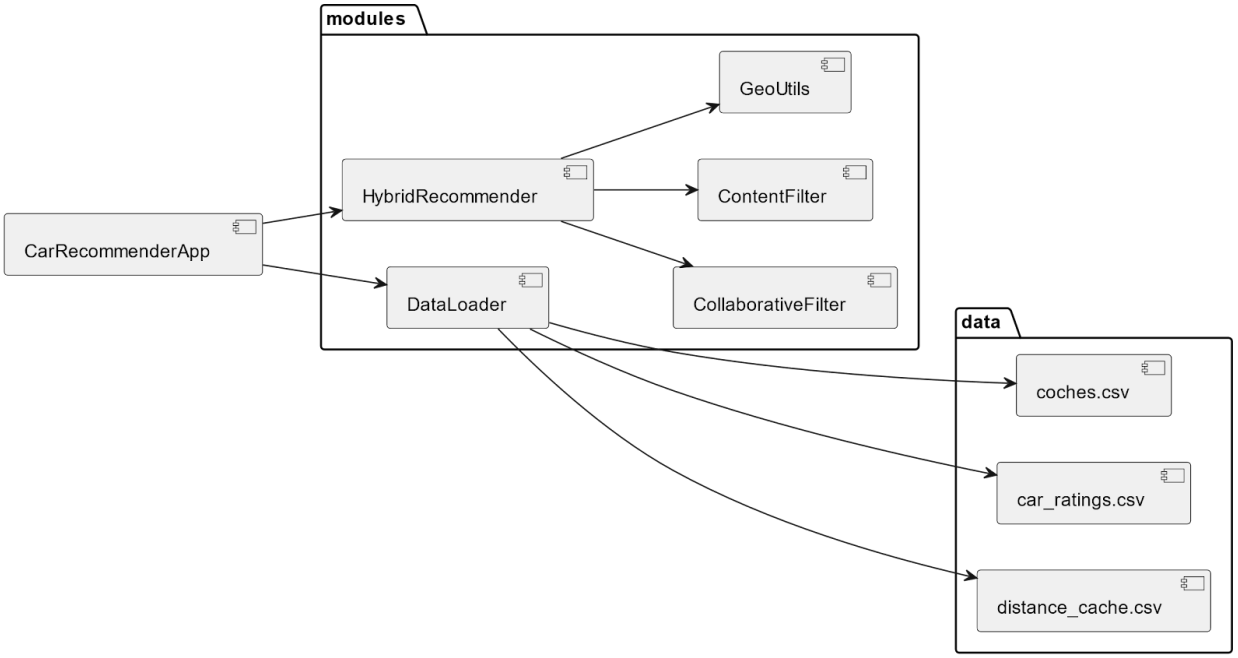


Diagrama de componentes



Pruebas

Validación y Limpieza del Dataset: Garantizar la calidad y consistencia de los datos utilizados para el entrenamiento y las recomendaciones.
Pruebas realizadas:

- Comprobación de valores faltantes y nulos en las columnas clave del dataset, como *power*, *color*.

Pruebas de Prototipo de GeoUtils con Geopy: Validar el funcionamiento de la biblioteca geopy y el cálculo de distancias entre ubicaciones. Se creó un script que tomaba dos ubicaciones (origen y destino) como entrada. Se utilizó geopy.geocoders.Nominatim para obtener coordenadas y geopy.distance.geodesic para calcular distancias.
Pruebas realizadas:

- Comparación de distancias calculadas con datos reales (e.g., distancia entre Madrid y Barcelona).

- Validación de la funcionalidad con ubicaciones no estándar (e.g., pueblos pequeños).

Pruebas de Similaridad de Contenido: Verificar si el sistema podía identificar correctamente similitudes entre coches basándose en sus características. Se creó un script que comparaba dos coches específicos utilizando métricas de similitud de características.
Pruebas realizadas:

- Comparación de coches con características muy similares para verificar que el sistema los consideraba "parecidos".

- Comparación de coches con características opuestas para asegurar que se identificaban como "diferentes".

Pruebas de Algoritmos de *Surprise*: Evaluar múltiples algoritmos de la biblioteca *Surprise* para identificar el modelo más eficiente en términos de precisión y capacidad predictiva, utilizando métricas estándar de error.

Método

- **Métricas de evaluación**:
    - **RMSE** (Root Mean Square Error): Evalúa el error promedio cuadrático de las predicciones.

    - **MAE** (Mean Absolute Error): Evalúa el error promedio absoluto de las predicciones.

    - **Tiempo**: Tiempo en segundos de lo que tarda en finalizar el entrenamiento cada algoritmo.

Ajuste de hiperparámetros: Una vez elegido el mejor algoritmo para nuestro trabajo (*SVD*), empleamos una búsqueda de los mejores hiperparámetros empleando *GridSearchCV* para ajustar aún mejor dicho algoritmo.

Manual

Manual de instalación

Requisitos previos

- **Python**: Asegúrate de tener Python 3 o superior instalado.
  Para verificar la versión instalada:

  ```
  python --version  # En Windows
  python3 --version  # En Linux
  ```

  Si no tienes Python, descárgalo desde python.org e instálalo (Importante, añadir Python al PATH).

- **Pip**: El gestor de paquetes de Python (generalmente incluido con Python).

  Verifica su instalación: `pip --version`

Pasos de instalación

Tras haber descargado el zip que contiene el programa, vamos a crear un entorno virtual para evitar conflictos con otras bibliotecas de Python.

Windows

```
python -m venv venv
venv\Scripts\activate
```

Linux

```
python3 -m venv venv
source venv/bin/activate
```

Ahora, dentro del entorno virtual y del directorio del programa, vamos a descargar todas las bibliotecas necesarias con el siguiente comando: `pip install -r requirements.txt`

Es posible que haya problemas a la hora de instalar Surprise, ya que esta librería requiere que instalemos Microsoft Visual C++ Build Tools y muy importante una versión de numpy<2 (ya indicado en el requirements.txt).

Para instalar el Microsoft Visual C++ Build Tools, basta con darle click al enlace que lleva a la instalación de estas dependencias. Es algo bastante intuitivo y está bien explicado en la página oficial de descarga.

Manual de uso

Una vez instalado todo, ya podemos ejecutar el programa de la siguiente manera:

Windows

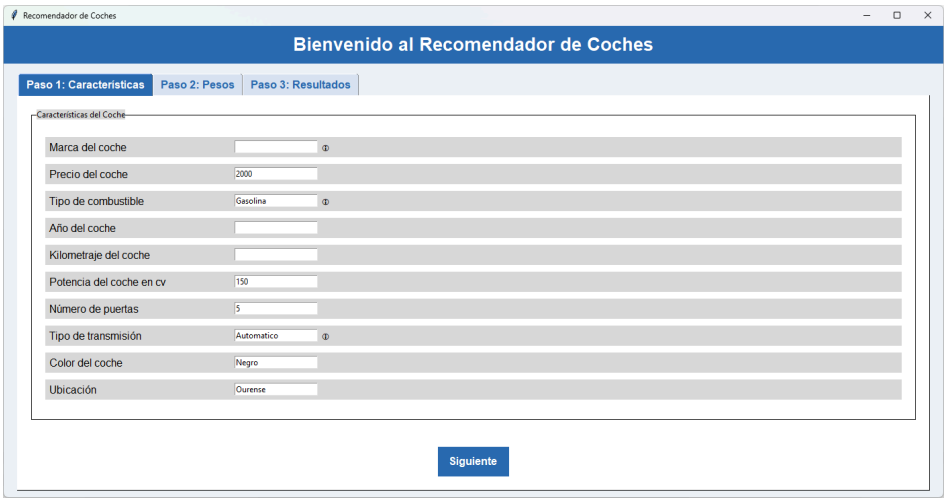
```
python car_recommender.py
```

Linux

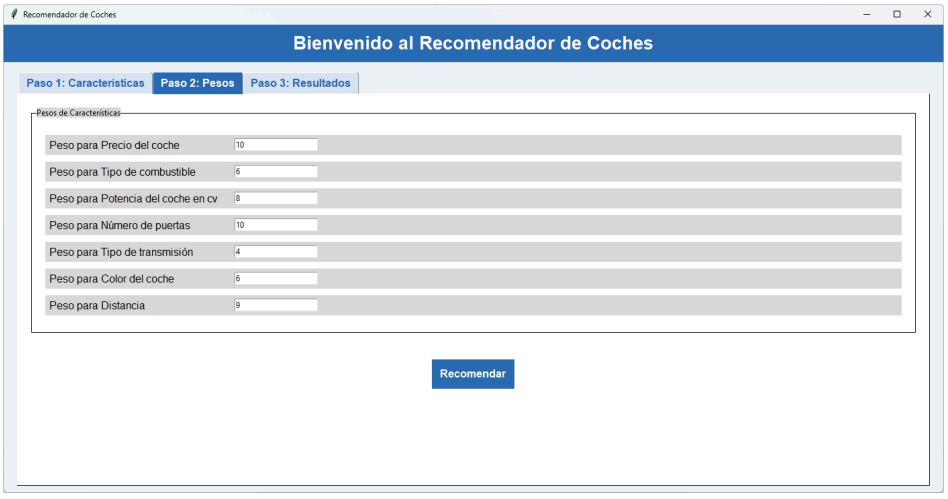
```
python3 car_recommender.py
```

Con este comando se mostrará la interfaz gráfica que es bastante intuitiva. A continuación se muestra un ejemplo de uso como modo de tutorial.

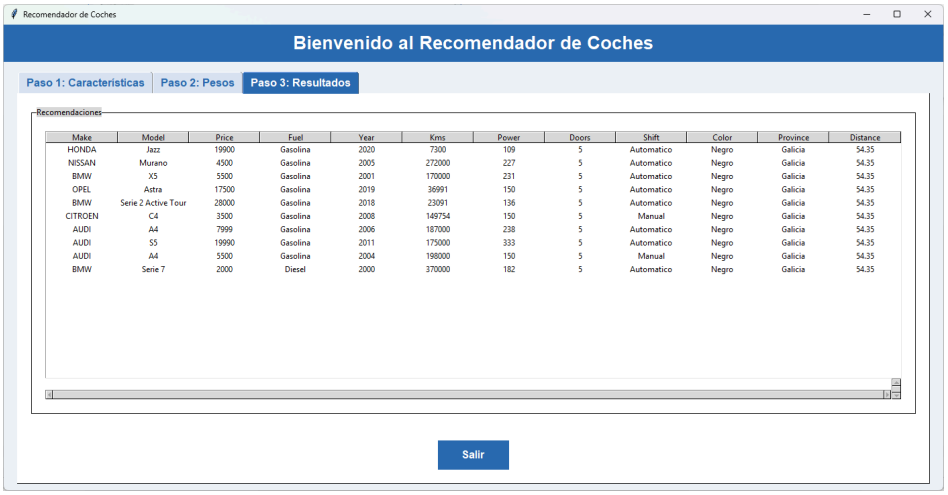
Primer paso: Hacer click en las casillas y escribir las preferencias. Si no sabe qué poner o no quiere poner algo en una o más casillas, se pueden dejar en blanco, pero hay que tener en cuenta que hay que poner como mínimo una característica y la ubicación del usuario es obligatoria. Darle al botón "Siguiente".



Segundo paso: Introducir un número del 1 al 10 la importancia que le da el usuario a cada una de las características que ha indicado en el primer paso. Darle al botón "Siguiente".



Tercer paso: Darle al botón que pone “Recomendar” y llevará a la última pestaña con los resultados indicados. Si el usuario desea usar de nuevo el programa debe de darle al botón “Salir” y ejecutar de nuevo como se mostró anteriormente y seguir los mismos pasos.



Nota: En el caso de que se haya puesto una ubicación que no se encuentre en el caché del programa, este procederá a calcularla y puede tardar unos 30 segundos en mostrar los resultados.

También existe la posibilidad de ejecutar el programa en una terminal si no se desea la versión gráfica. El funcionamiento de introducción de los datos sigue la misma lógica y se ejecuta de la siguiente manera:

Windows

```
python car_recommender_cli.py
```

Linux

```
python3 car_recommender_cli.py
```

Resultados

Rendimiento y Métricas

Los diferentes algoritmos nos dieron los siguientes resultados:

Algoritmo	RSME	MAE	Tiempo (s)
BaselineOnly	0,5169	0,4954	10,63
SVD++	0,5198	0,4950	61,18
SVD	0,5215	0,4950	13,83
KNNBasic	0,5219	0,4954	355,75
NMF	0,5220	0,4946	15,04
KNNBaseline	0,5222	0,4968	365,86
CoClustering	0,5227	0,4951	17,76
KNNWithMeans	0,5233	0,4971	359,92

Aunque ***BaselineOnly*** presentó un rendimiento ligeramente superior en las métricas *RMSE, MAE* y *Tiempo*, su enfoque se limita al uso de promedios generales, tanto de usuarios como de ítems, sin modelar las interacciones más profundas entre ambos. Esta limitación es significativa en nuestro caso, ya que los ratings están segmentados por categorías como deportivos, lujo o económicos, donde entender y capturar relaciones complejas entre usuarios y modelos de coches es esencial para generar recomendaciones personalizadas y precisas. Por esta razón, a pesar de sus buenos resultados, ***BaselineOnly*** no satisface plenamente nuestras necesidades.

El resto de los algoritmos fueron descartados principalmente debido a los elevados tiempos requeridos para su entrenamiento, un factor crítico dado el tamaño de nuestro dataset, que supera el millón de registros. Métodos como ***KNNBasic***, ***KNNBaseline*** y ***KNNWithMeans*** realizan cálculos intensivos de similitud, lo que los hace ineficientes en conjuntos de datos tan grandes. Algoritmos como ***SVD++*** y ***CoClustering***, aunque competitivos en precisión, introducen una complejidad adicional que incrementa significativamente los tiempos de entrenamiento, haciéndolos poco prácticos para este proyecto.

En contraste, ***SVD*** destaca como la solución más adecuada, ya que combina una precisión sólida con tiempos de entrenamiento razonables. Este algoritmo no solo captura las relaciones subyacentes entre usuarios y coches de forma efectiva, sino que también optimiza el proceso al reducir la dimensionalidad del problema. Esto convierte al ***SVD*** en la mejor opción para satisfacer los requerimientos del proyecto.

Tiempos de ejecución

El tiempo medio de entrenamiento del modelo se aproxima a los 13,83 segundos, dependiendo de la máquina en la que se ejecute.
Por otra parte, al utilizar la librería geopy, si no se introduce una ciudad ya guardada en el cache, tiene que llamar a la API de Google lo que puede llevar bastante tiempo, generalmente no más de 30 segundos. De nuevo, esto depende de la máquina que se esté usando.

Discusión

Limitaciones

El sistema desarrollado no abarca aspectos transaccionales, como la compra directa del coche, la negociación de precios o los procedimientos legales relacionados con la adquisición del vehículo.Sin embargo, utilizamos datos obtenidos de coches.net lo que permite al usuario buscar directamente los anuncios en dicha plataforma para continuar con el proceso de compra si lo desea.

Dado que el dataset utilizado proviene de una web específica, no incluye todas las opciones disponibles en el mercado global. A pesar de esta limitación, el sistema ha sido diseñado para ser escalable, por lo que basta con proporcionar un dataset más amplio, que contenga un mayor número de anuncios, modelos y marcas, siempre que mantenga los mismos parámetros para cada instancia. Esto permitiría obtener una solución más completa y recomendaciones de mayor alcance.

Por último, cabe destacar que el sistema proporciona una **guía orientativa** basada en las preferencias del usuario y las valoraciones de otros compradores. Por lo tanto, las recomendaciones generadas no deben considerarse una decisión definitiva, sino un apoyo para facilitar el proceso de selección de vehículos.

Mejoras

Una posible mejora futura sería implementar nuestra interfaz en una página web, lo que facilitaría enormemente su uso para los usuarios. A través de esta plataforma, los usuarios podrían crear una cuenta personal, lo que les permitiría acceder a un historial de sus recomendaciones.

Además, esta implementación web simplificaría la gestión del sistema, ya que permitiría actualizar el dataset de forma centralizada sin que los usuarios tengan que realizar descargas manuales. Esto aseguraría que las recomendaciones se basen siempre en los datos más recientes y relevantes, mejorando así la experiencia del usuario y la precisión del sistema.

Problemas encontrados

El problema principal con el que tuvimos que lidiar fue la librería *Surprise*. Esto se deba a que se basa en recomendaciones hechas por otros usuarios (sistema de recomendación **colaborativo**) y es complicado obtener esta información en el caso de los vehículos ya que una persona común no compra coches diariamente como para poner reviews de forma frecuente.

Para solucionar esto, hemos creado un dataset (*car_ratings.csv*) con un script en python donde:

- Hemos indicado que hay 15.000 usuarios que van a dar muchas valoraciones cada uno.
- Hemos separado los coches en segmentos (lujosos, económicos, deportivos...) según las marcas con unas probabilidades asociadas a cada uno.

Obteniendo un dataset con +1.000.000 de instancias (valoraciones).

Debido a que hemos separado los coches en segmentos, obtenemos un dataset balanceado en clases representativas, lo que le viene genial al algoritmo *SVD*, que busca patrones ocultos en los datos para una mejor recomendación.

Bibliografía

https://surprise.readthedocs.io/en/stable/
https://geopy.readthedocs.io/en/stable/
https://github.com/Data-Market/vehiculos-de-segunda-mano