Sistemas Inteligentes
Grupo 4:
Sistema Recomendador

Eduardo Suárez Ojeda (alu0100896565)
Florentín Pérez González (alu0101100654)
Adrián Emilio Padilla Rojas(alu0101138558)

Índice:

Introducción:

El objetivo de nuestro proyecto es el de crear un sistema recomendador para todo tipo de objetos, es decir, con un catálogo infinito.

Para su realización hemos optado por crear una aplicación web que permite, mediante registro de usuario, realizar búsquedas de ítems por todo tipo de webs con una sola búsqueda y mostrar los resultados en nuestra página, pudiendo añadir los resultados a la lista de favoritos del usuario.

La aplicación también posee una interfaz para acceder a las recomendaciones para el usuario cuya sesión está activa. Para realizar estas recomendaciones se obtienen valoraciones de los usuarios de manera implícita, como por ejemplo los datos de los objetos que busca el usuario en nuestro buscador o su lista de favoritos.

Existen distintos tipos de recomendaciones:

- Recomendación general: En la página de inicio de la aplicación se mostrarán ítems sin que el usuario realice una búsqueda previa sobre los mismos. Dichos objetos se obtienen a partir de la página de últimas novedades de Amazon España. No requiere de ningún tipo de información sobre el usuario y se podría catalogar en un sistema recomendador por popularidad.

- Recomendación individual: Basada únicamente en las valoraciones del propio usuario. Este tipo de recomendación se basa en calcular las N categorías con mayor valoración por parte del usuario y mostrar ítems de dichas categorías que se encuentren en la sección de Novedades Destacadas de la página de Amazon de dicha categoría. Se trata de un sistema que va un poco más allá que el anterior, pero no es demasiado profundo, ya que se basa en únicamente los datos del propio usuario y las recomendaciones son los mismos ítems para todos los usuarios de cada categoría, aunque a lo largo del tiempo se van actualizando (ya que se obtienen de la web de Amazon y estos se modifican a lo largo del tiempo). Se trata de un sistema de recomendación basado en contenido, ya que se recomienda únicamente teniendo en cuenta el perfil del usuario actual.

- Recomendación por K vecinos: Se trata de un tipo de recomendación más complejo que los anteriores, ya que tiene en cuenta el conjunto de usuarios para generar recomendaciones. Se trata de aplicar un algoritmo que localice los usuarios con un perfil de valoraciones muy similar al del usuario activo y realizar las recomendaciones basándose en sus listas de favoritos. Se realiza sobre la matriz de valoraciones global de todos los usuarios y se realiza un algoritmo KNN con los parámetros elegidos por los administradores. Después de muchas pruebas se ha optado por un algoritmo KNN basado en la puntuación-z de cada usuario, que proyecta la representación multidimensional de los usuarios a un único eje y selecciona los más próximos entre sí. Dado que nuestra base de datos de usuarios es relativamente pequeña optamos por entrenar el sistema con toda la base de datos y de realizar una validación cruzada para evaluar los resultados. También

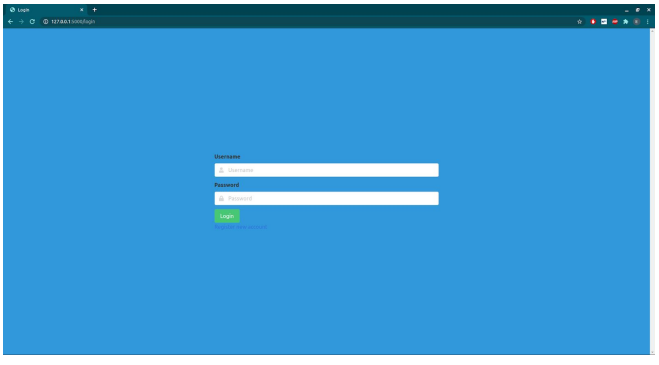
hemos optado por elegir las opciones de medida ‘MSD’ y ‘FCP’ para obtener los mejores resultados. Con dichas medidas hemos aumentado de tener una media de acierto de 0,20 (con parámetros diferentes) a llegar a 0,7 o superior con una media de 0,65. Se trata de un sistema recomendador colaborativo, ya que usa las valoraciones del conjunto de usuarios y relaciona los usuarios similares. Un punto positivo de este tipo de recomendación es que permite recomendar ítems para los que el usuario a priori no tiene apenas valoraciones (para su categoría) pero que por tenerlos un usuario similar pueden ser de su agrado e incluso que se le descubra un nuevo gusto, lo cual es un factor importante.

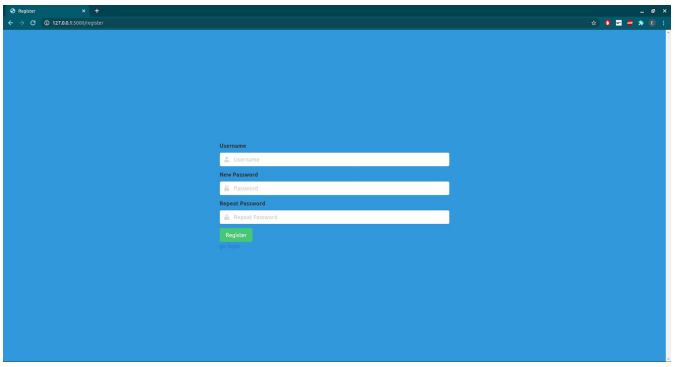
- Recomendación por SVD y SVDpp: Utilizamos estos algoritmos para obtener recomendaciones para cada usuario sin necesidad de recurrir al filtrado colaborativo. Se tratan de famosos algoritmos matriciales que permiten definir a un usuario en función de sus intereses. A tal fin, y como se comentará más adelante en otra sección, se encuentra implementado un sistema interno que realiza la gestión de dicho parámetro. Ambos algoritmos nos permitirán obtener, de las categorías que el usuario no considera, aquellas más prometedoras para a través del componente de “web scrapping”, obtener los objetos más populares de estas. Al igual que los algoritmos KNN, estos poseen parámetros que definen su comportamiento y que deben ser ajustados para obtener los mejores resultados posibles. La obtención de estos resultó sencilla, pues se utilizó un mecanismo de fuerza bruta para hallar los valores más adecuados para algunos de los parámetros. En concreto, los parámetros considerados son el ratio de aprendizaje (0.09 y 0.01 respectivamente) y el número de iteraciones del procedimiento SGD adherente al funcionamiento de estos algoritmos de descomposición matricial (30 y 10 respectivamente). Para ambos casos, se ha empleado “FCP” como opción de medida, obteniendo resultados que oscilan entre 0.56 a 0.7.

Interfaz:

Página de Login/Registro:

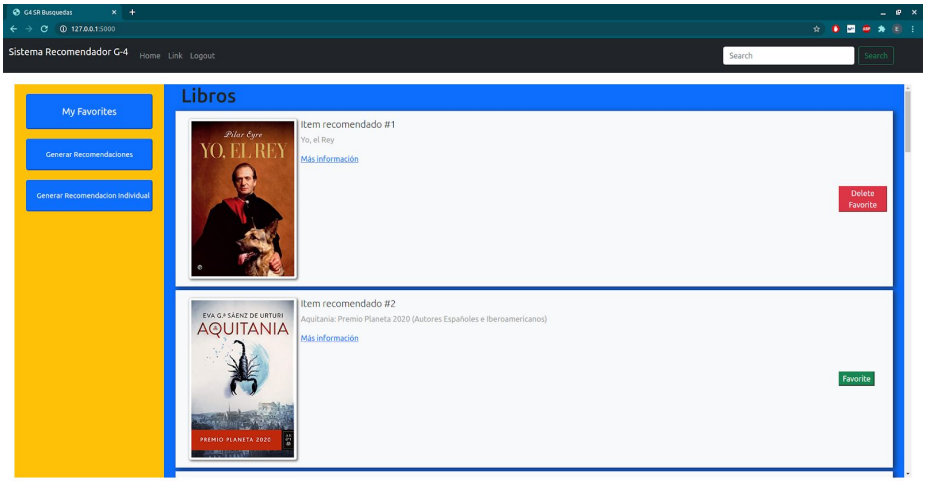
Interfaz muy sencilla de login/registro pero totalmente necesaria para acceder a la web, ya que si no sería imposible establecer recomendaciones para usuarios anónimos, a la vez que cuanto mayor sea el número de usuarios, más precisas serán nuestras recomendaciones basadas en sistemas colaborativos.





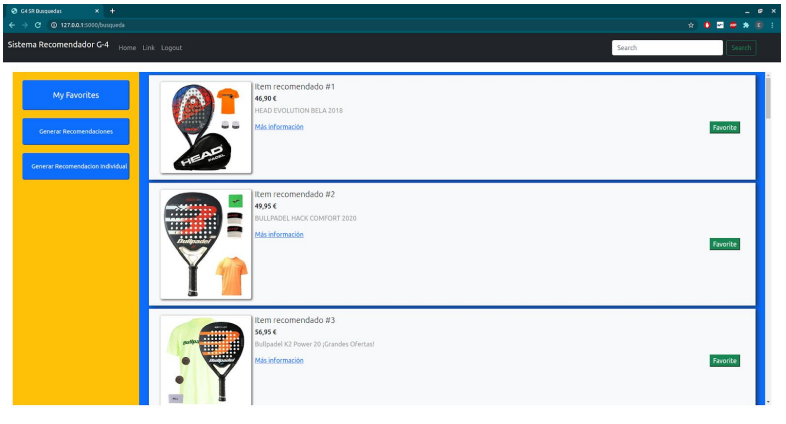
Página Home:

Se trata de una página donde se muestran recomendaciones generales basadas en popularidad obtenidas de las Últimas Novedades de Amazon España.



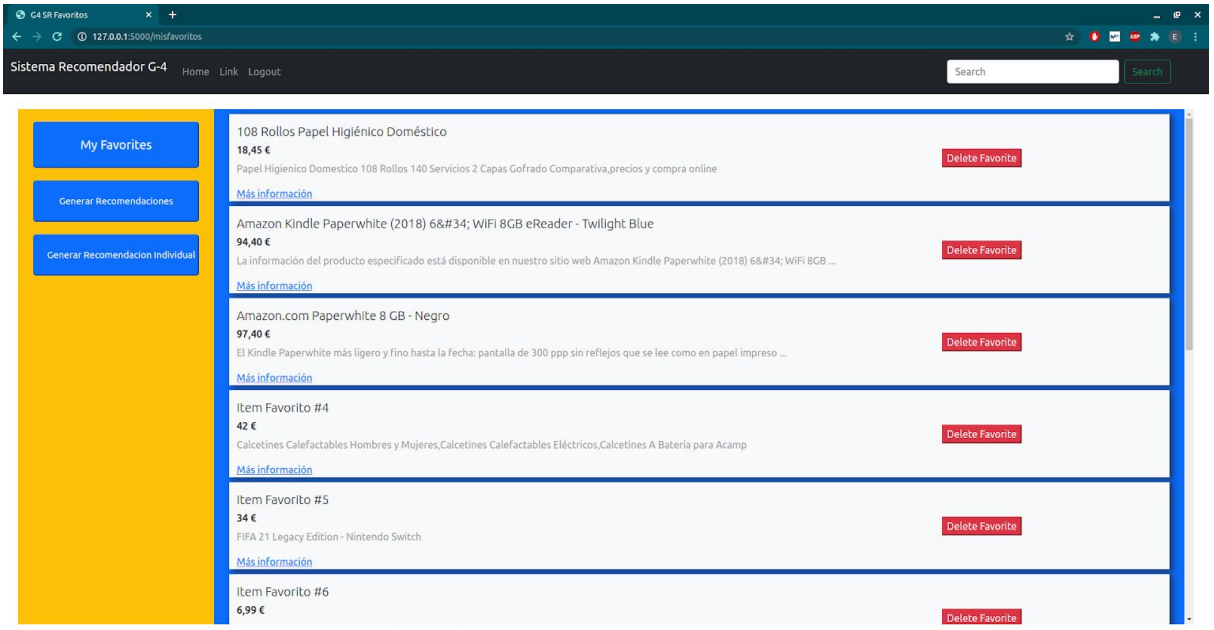
Página de búsqueda:

Se muestran los diferentes resultados obtenidos a partir de la búsqueda escrita en la sección Shopping de Google, con lo que los resultados se obtienen de todo tipo de webs de tiendas en la red.



Página de Favoritos:

Donde se guarda la lista de favoritos del usuario actual.



Los ítems no tienen foto porque los “srcs” de las imágenes son temporales y guardar el obtenido al realizar la búsqueda no te garantiza tenerlo para más adelante, pero sí se guardan los datos y la web de la tienda que ofrece el producto.

Páginas de recomendaciones:

Son similares a las páginas de búsquedas pero con los ítems obtenidos al aplicar los sistemas de recomendación al usuario actual.

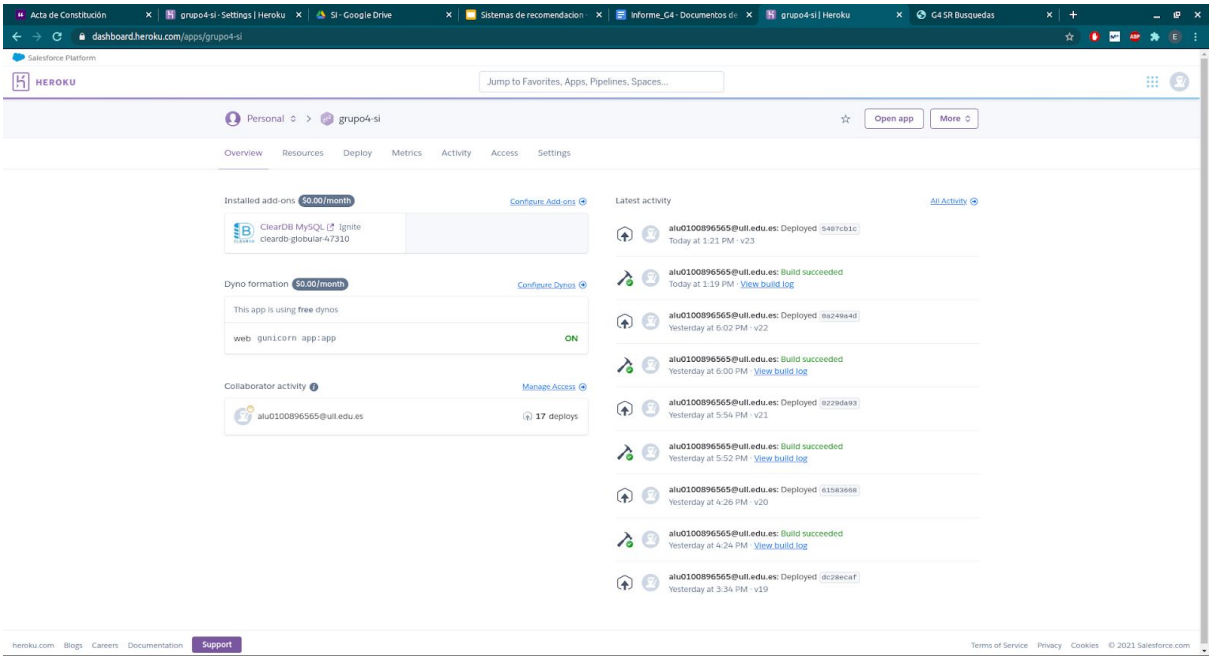
Estructura de la aplicación:

Dado que los sistemas inteligentes se realizan con el lenguaje ***Python*** hemos optado por realizar todas las partes de la aplicación con dicho lenguaje para facilitar la integración de los diferentes módulos de la aplicación.

Hemos usado ***Flask*** para el desarrollo de la web, ***Selenium*** para la parte de web Scapping; y ***Pandas*** y ***Surprise*** para el sistema recomendador. Dentro de la propia web se usa ***HTML***, ***CSS***, ***JavaScript*** (con ***jQuery*** para realizar peticiones Ajax al backend de Python en Flask) y ***jinja2*** para generar HTML dinámico. La base de datos se encuentra en el servidor de nuestro proveedor web y usa el lenguaje ***MySQL***.

Dado que nuestro sistema depende altamente de la información que dispongamos en nuestra base de datos es crucial tener la mayor cantidad de usuarios posibles para mejorar las recomendaciones. Por ello hemos optado por un despliegue de la aplicación web en la plataforma ***Heroku***. Dicha plataforma permite desplegar aplicaciones como la nuestra, basada en Python, de forma muy sencilla. También es donde hemos añadido la base de datos como add-on. El problema es que al optar por un plan gratuito poseemos algunas limitaciones de tamaño: 500 MB para el código y todo lo que sea necesario descargar (librerías Python principalmente) y 5 MB para la base de datos. Dichas limitaciones no nos han afectado por ahora, ya que a fecha de redacción de este documento tenemos 250 MB de código y menos de 1 MB en la base de datos.

La web se puede acceder con el siguiente enlace para usuarios externos. Aunque acceder por medio de la web en vez de su versión local tiene la desventaja de que posee unos tiempos de carga bastante largos debido principalmente a la parte de Web Scrapping que se usa en casi todas las acciones de la web.



Funciones de Web Scrapping:

Usamos ***Selenium*** con ***Chromedriver*** para automatizar todas estas funciones. La primera de las funciones que desarrollamos corresponde a la que se encarga de realizar la búsqueda de ítems cuando el usuario realiza una búsqueda en la aplicación web. Esta función se dirige a la página de "Google Shopping" y realiza la misma búsqueda que aquella concretada originalmente en el buscador de la web. Se consideran todos los resultados obtenidos y se van guardando en una lista como diccionarios que contiene:

- Nombre
- Descripción
- "Src" de la foto
- Link de la página de compra
- Precio

El principal problema es que existen dos tipos de resultados, cuando se buscan objetos más generales, como en el caso de la ropa, los ítems no poseen un nombre, sino la descripción del producto. Por lo tanto nuestra función realiza dos iteraciones por los objetos con el fin de cubrir ambos casos.

```
try:
    driver.get('https://www.google.es/search?q=' + busqueda + '&tbm=shop')

    # Objeto que guarda el array de items para pasar a JSON
    items = []

    # ps5
    aa = driver.find_elements_by_css_selector('div.sh-dlr__list-result')

    for item in aa:
        itemObj = {}
        itemObj['fotoSrc'] = item.find_element_by_class_name('TL92Hc').get_attribute('src')
        itemObj['name'] = item.find_element_by_class_name('xsRiS').text
        itemObj['price'] = item.find_element_by_class_name('O8U6h').text
        itemObj['price'] = itemObj['price'].split('\n')[0]
        posibleDescripcion = item.find_elements_by_class_name('hBUZL')
        if len(posibleDescripcion) == 2:
            itemObj['description'] = posibleDescripcion[1].text
        else:
            itemObj['description'] = posibleDescripcion[2].text
        itemObj['linkGS'] = item.find_element_by_class_name('FkMp').get_attribute('href')
        items.append(itemObj)

    # botas
    bb = driver.find_elements_by_css_selector('div.sh-dgr__grid-result')

    for item in bb:
        itemObj = {}
        itemObj['name'] = 'NoName'
        itemObj['fotoSrc'] = item.find_element_by_class_name('MUQY0').find_element_by_tag_name('img').get_attribute('src')
        itemObj['price'] = item.find_element_by_class_name('kHxwFf').text
        itemObj['price'] = itemObj['price'].split('\n')[0]
        itemObj['description'] = item.find_element_by_class_name('A2sOrd').text
        itemObj['linkGS'] = item.find_element_by_class_name('ty2Wqb').get_attribute('href')
        items.append(itemObj)
    tipo = amazCateg(itemName)
finally:
    driver.close()
```

A su vez dentro de la propia función se llama a otra (amazCateg) para obtener el tipo de los objetos de la búsqueda con el fin de asignarlos a una de las categorías que poseemos para diferenciar los objetos entre sí.

Dado que dichas categorías se buscan en la página web de Amazon, contamos con algo más de 40 categorías, las mismas que posee dicha página. La función que se llama para detectar el tipo de los objetos en las búsquedas se usa para establecer dicha categoría. Únicamente se le pasa como argumento el nombre del item que se quiere categorizar, se realiza la búsqueda en la página de Amazon España y se obtiene la categoría de Amazon a la que pertenece. Esta función se llama también a la hora de añadir objetos a los favoritos de cada usuario por si la elección es de una categoría diferente a la que se ha obtenido primeramente al realizar la búsqueda, llamando a la función con el nombre del item seleccionado como parámetro.

```python
def amazCateg(itemName):
    # chrome_options = webdriver.ChromeOptions()
    # chrome_options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")
    # chrome_options.add_argument("--headless")
    # chrome_options.add_argument("--disable-dev-shm-usage")
    # chrome_options.add_argument("--no-sandbox")
    # driver = webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH"), chrome_options=chrome_options)
    driver = webdriver.Chrome() # usar en local
    driver.get('https://www.amazon.es/')
    inputB = driver.find_element_by_id('twotabsearchtextbox')
    inputB.send_keys(itemName)
    inputA = driver.find_element_by_id('nav-search-submit-text')
    actions = ActionChains(driver)
    actions.move_to_element(inputA).perform()
    actions.click().perform()
    try:
        inputC = WebDriverWait(driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'h2 a')))
    except:
        print("Loading took too much time!")
        driver.close()
        return False
    aa = inputC.get_attribute('href')
    driver.get(aa)
    try:
        WebDriverWait(driver, 2).until(EC.presence_of_element_located((By.XPATH, "//*[contains(text(), '€')]")))
        txt = driver.find_elements_by_xpath("//*[contains(text(), 'n°')]")
        if len(txt) == 2:
            textt = txt[0].text
        else:
            txt = driver.find_element_by_xpath("//*[contains(text(), 'Clasificación en los más vendidos de Amazon:')]").find_element_by_
            textt = txt.text
        driver.close()
        return REGEX.match(textt)[1].strip()
    except:
        driver.close()
        return False
```

La tercera función (amazonRecommend) es la que se encarga de obtener las recomendaciones individuales de la página. Se le pasa como argumento la lista de las categorías mejor valoradas por el usuario y obtiene las *Novedades Destacadas* en la página de Amazon para cada categoría.

Accede a las páginas principales de cada una de las categorías y elige N ítems de cada una de ellas. En la propia función, N se va reduciendo a medida que se llega a categorías con menores valoraciones para el usuario. Debido a que nos encontramos en la propia web de Amazon el tipo de los ítems es el mismo que el de la categoría en la que nos encontremos. También en la propia función encontramos varios bloques “try” “except” debido a que en las diferentes categorías no siempre se encuentra la sección de novedades destacadas y necesitamos tratar con estas situaciones para que no se produzcan fallos en la página y poder seguir obteniendo recomendaciones válidas.

```python
    for item in favCategories:
        if item[1] == 0:
            continue
        driver.get('https://www.amazon.es/')
        try:
            WebDriverWait(driver, 2).until(EC.presence_of_element_located((By.ID, 'searchDropdownBox')))
        except:
            print("Loading took too much time!")
        categories = driver.find_element_by_id('searchDropdownBox')
        for option in categories.find_elements_by_tag_name('option'):
            if option.text == item[0]:
                option.click() # select() in earlier versions of webdriver
                actions = ActionChains(driver)
                inputA = driver.find_element_by_id('nav-search-submit-text')
                actions.move_to_element(inputA).perform()
                actions.click().perform()
                break
        time.sleep(0.8)
        try:
            novedades = driver.find_element_by_xpath("//*[contains(text(), 'Novedades destacadas')]").find_element_by_xpath('..').fin
        except:
            novedades = driver.find_element_by_xpath("//*[contains(text(), 'Los más vendidos')]").find_element_by_xpath('..')
        try:
            amzItems = novedades.find_element_by_tag_name('ul').find_elements_by_tag_name("li")
        except:
            amzItems = novedades.find_element_by_tag_name('ol').find_elements_by_tag_name("li")
        for i in range(busq):
            itemCar = {}
            info = amzItems[i].find_element_by_tag_name('a')
            itemCar['name'] = 'NoName'
            itemCar['description'] = amzItems[i].find_element_by_tag_name('img').get_attribute('alt')
            try:
                itemCar['price'] = amzItems[i].find_element_by_class_name('a-price-whole').text + ' €'
            except:
                continue
            itemCar['linkGS'] = info.get_attribute('href')
            itemCar['fotoSrc'] = amzItems[i].find_element_by_tag_name('img').get_attribute('src')
            itemCar['tipo'] = item[0]
            resultado.append(itemCar)
        busq = 1
```

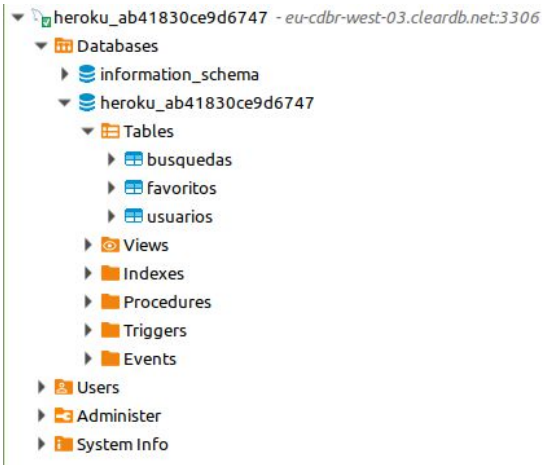
La cuarta función (homeScrap) se encarga de rellenar de ítems nuestra página de Home. Para ello accedemos una vez más a Amazon España y a la sección de Últimas Novedades, donde se muestran ítems de categorías variadas (van cambiando incluso en búsquedas con pocos segundos de diferencia) y generalmente populares para rellenar nuestra página de inicio.

```python
def homeScrap():
    # chrome_options = webdriver.ChromeOptions()
    # chrome_options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")
    # chrome_options.add_argument("--headless")
    # chrome_options.add_argument("--disable-dev-shm-usage")
    # chrome_options.add_argument("--no-sandbox")
    # driver = webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH"), chrome_options=chrome_options)
    driver = webdriver.Chrome() # usar en local
    driver.get('https://www.amazon.es/')
    mlvHref = driver.find_elements_by_xpath("//*[contains(text(), 'Últimas Novedades')]")[0].get_attribute('href')
    driver.get(mlvHref)
    secciones = {}
    try:
        WebDriverWait(driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'div h3')))
        sections = driver.find_elements_by_class_name('zg_homeWidget')
        for section in sections:
            nameSect = section.find_element_by_tag_name('h3').text
            secciones[nameSect] = []
            imgs = section.find_elements_by_css_selector('div a div img')
            for img in imgs:
                item = {}
                item['name'] = 'NoName'
                item['description'] = img.get_attribute('alt')
                item['linkGS'] = img.find_element_by_xpath('..').find_element_by_xpath('..').get_attribute('href')
                item['fotoSrc'] = img.get_attribute('src')
                item['tipo'] = nameSect
                secciones[nameSect].append(item)
    finally:
        driver.close()
        return secciones
```

La quinta función (getPriceAmaz) sirve únicamente para obtener la información del precio de los objetos de la página principal de la web, ya que al buscarlos con la anterior función no se muestra el precio de ellos en la página de Amazon. La función se utiliza cuando uno de estos productos es añadido a los favoritos de los usuarios para obtener la información del precio y guardarlo en la base de datos.

Base de datos:

La estructura de nuestra base de datos es muy simple, en parte condicionada por el poco espacio que tenemos para alojar los datos de manera gratuita (5 MB). Contamos con una tabla para almacenar los usuarios donde se guarda el nombre de usuario y su contraseña. En otra tabla se guardan los datos de las búsquedas de los usuarios para obtener las valoraciones en forma de búsqueda de dichos usuarios. Por último, la tabla de favoritos en la que se guardan los ítems pertenecientes a las listas de favoritos de todos los usuarios.



Se han establecido tablas generales (y no una por cada usuario, como se intentó al inicio) para ahorrar en espacio en la base de datos y para simplificar las búsquedas, debido además a que para obtener la matriz de valoraciones nos basta con una búsqueda por tabla. En las tablas de favoritos y búsquedas se tiene el atributo “username” como clave ajena del atributo del mismo nombre en la tabla usuarios.
Las claves primarias de las tablas son:

- Usuarios: username
- Favoritos: name, description, price
- Búsquedas: ID (auto incremental)

Funciones de obtención de valoraciones:

Disponemos de varias funciones para obtener las valoraciones de los usuarios, dependiendo del tipo de valoración que se va a realizar.

Para las recomendaciones individuales tenemos la función *getValuser*, a la que se le aportan los argumentos de las filas de favoritos y búsquedas relacionadas con el usuario actual. Se realiza la suma de valoraciones para las distintas categorías en un diccionario que las contiene a todas y las inicializa a 0. Para obtener la suma se le da un valor a las valoraciones por medio de búsquedas (de 1 en nuestro caso) y otro a las de favoritos (5 en concreto). Una vez realizado el cálculo total, ordenamos el diccionario por los valores numéricos y retornamos una lista con las N categorías más valoradas por el usuario junto con el número de valoraciones de cada una de ellas en forma de tuplas ((nombre, valTotal)).

Esta función es la que llama la función de Web Scrapping *amazonRecommend* para obtener las recomendaciones de popularidad basadas en las categorías favoritas del usuario.

También tenemos la función *getPandas*, que devuelve un dataframe de pandas con tres columnas: usuario, categoría y valoración. Dicho dataframe se usa para la función de recomendación colaborativa por medio del algoritmo KNN. Se le pasan como argumentos el contenido de las tres tablas de la base de datos y de manera similar a la anterior función se genera la suma total de las valoraciones. La diferencia es que en este caso, ya que las funciones de recomendación funcionan mejor cuando se tienen valores de mínimo y máximo para las valoraciones lo que se hace es dividir todas las valoraciones por categoría de cada usuario por el valor de la categoría con mayor valoración, con lo que los resultados siempre se encontrarán entre 0 y 1, mejorando el resultado de los algoritmos.

Funciones de algoritmos de recomendación:

En este caso tenemos las funciones de recomendaciones por medio de KNN. La función *getRecomendations* se lanza con el dataframe de pandas con las valoraciones de los usuarios a las categorías, lo convierte en un dataset válido para usar con la librería *Surprise* y se establecen los parámetros y las distintas opciones que posee el algoritmo

para obtener la mayor precisión y los mejores resultados posibles. Hemos separado en dos funciones el código debido a que la primera es la encargada de entrenar el sistema y pensábamos que iba a ser muy costosa a nivel temporal, pero con una base de usuario como la que tenemos actualmente se ejecuta en menos de 1 segundo, pero queremos mantenerlas divididas para mejorar su rendimiento en el futuro.

```
124
125  def getRecomendations(dataF):
126    global algo, trainSet
127    reader=Reader(rating_scale=(0., 1.))
128    data = Dataset.load_from_df(dataF, reader)
129    trainSet = data.build_full_trainset()
130    k = 5
131    minimunK = 1
132    options = {
133      'name': 'MSD',
134      'user_based': False,
135    }
136    # algo = KNNBasic(k = k, min_k = minimunK, sim_option = options)
137    # algo = KNNWithMeans(k = k, min_k = minimunK, sim_option = options)
138    algo = KNNWithZScore(k = k, min_k = minimunK, sim_option = options)
139    algo.fit(trainSet)
140
141    results = cross_validate( # Tarda en finalizar. Se ejecuta unas 3 veces por cv=3
142      algo=algo, data=data, measures=['FCP'],
143      cv=3, return_train_measures=True
144      )
145    print(results['test_fcp'].mean())
146
147  def getRecomUser(username):
148    global algo, trainSet
149    idUser = trainSet.to_inner_uid(username)
150    listVecinosBien = [trainSet.to_raw_uid(x) for x in algo.get_neighbors(idUser, 5)]
151    print(listVecinosBien)
152    return listVecinosBien
```

La segunda función *getRecomUser*, únicamente se encarga de traducir los resultados obtenidos a los nombres de usuario de los K vecinos, en forma de lista, del usuario que se pasa como argumento.

La función que, con base en los datos obtenidos mediante el uso de estas funciones de recomendación organiza los ítems a recomendar al usuario es la función *getItemsFromUsers*, que, como su nombre indica, su propósito es el de generar los items a recomendar para el usuario en base a sus vecinos más próximos.

```
def getItemFromUsers(favoritos, users, userActual):
    itemsRecomendacion = []
    simplifList = ['name', 'description', 'price']
    # Items del usuario para no recomendarle objetos que ya tiene en su lista
    itemsUserActual = [{key: value for (key, value) in x.items() if key != 'usuario'} for x in favoritos if x['usuario'] == userActual]
    userItemSimplif = [{key: value for (key, value) in x.items() if key in simplifList} for x in itemsUserActual]
    # Matriz de items de los usuarios similares
    matrixRecomendations = [[{key: value for (key, value) in x.items() if key != 'usuario'} for x in favoritos if x['usuario'] == user] for user in users]
    for idx, listItem in enumerate(matrixRecomendations):
        # Eliminar los ítems que el usuario ya tenga en su lista de favoritos
        listItem = [item for item in listItem if {'name': item['name'], 'description': item['description'], 'price': item['price']} not in userItemSimplif]
        # Ordena aleatoriamente la lista de favoritos del usuario vecino para recomendar ítems aleatorios de su lista
        random.shuffle(listItem)
        # Obtengo más ítems de los vecinos más próximos
        itemsRecomendacion += listItem[:(5 - idx)]
    # Añade la cualidad de no ser favorito del usuario
    for item in itemsRecomendacion:
        item['fav'] = 'nofav'
    return itemsRecomendacion
```

La función básicamente trata de seleccionar entre las listas de favoritos de sus vecinos más cercanos los items que se le van a recomendar al usuario activo, teniendo en

cuenta que no pertenezcan ya a su lista y de manera aleatoria entre las listas de sus vecinos, priorizando a los más cercanos.

Diferencias con el ACP:

La principal diferencia con respecto al acta de constitución de proyecto reside en la funcionalidad de hacer un tracking a los precios de los diferentes ítems. Este hecho se debe a que al buscar los productos en todo tipo de páginas no podemos establecer una función en Web Scrapping para actualizar los precios de los productos porque no existe una clase de elemento HTML, un ID o simplemente texto que buscar de manera similar en todas esas webs, por lo que nos parece imposible llegar a obtener esta funcionalidad sin que sea muy costosa de implementar y dado que el trabajo se basa más en obtener el sistema recomendador la hemos desestimado por su alta dificultad con respecto a la poca relevancia que tendría en el proyecto.

Conclusiones:

Como conclusión principal queríamos recalcar la importancia que tiene para nosotros los estudiantes el tener libertad para elegir el proyecto que queramos realizar a la hora de valorar la motivación que obtenemos por ello y por poder realizar un proyecto que a priori nos parece interesante como en este caso.

Por otro lado, centrándonos en el propio proyecto hemos aprendido mucho sobre los sistemas de recomendación que están tan a la orden del día últimamente con la cantidad de datos que se tiene en las empresas tecnológicas más punteras sobre sus usuarios y lo que se puede llegar a realizar con dichos datos. Hoy en día, cuando la publicidad está tan optimizada para que la reciba un target en principio más afín a los productos que se ofrecen y con ello se produzca un ahorro y también un aumento de las ventas al mejorar la precisión.

Dado que hemos sido nosotros mismos los encargados de rellenar los usuarios de la base de datos y sus gustos, el sistema de recomendación que se encarga de relacionar usuarios ya sabíamos que no nos iba a generar recomendaciones interesantes para nosotros, sin embargo, nuestro recomendador individual sí que ha sido capaz de mostrarnos nuevos objetos interesantes para tener en cuenta.

También hemos aprendido mucho sobre realizar un deploy de una web (en nuestro caso en Heroku) que se basa en un sistema recomendador y totalmente programada en Python, y en nuestro caso concreto, a realizar un full stack uniendo muchos módulos diferentes, lo cual ha sido una experiencia valiosísima para nosotros.