Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Máster Profesional en Ingeniería Informática

Curso 2020/2021

PRÁCTICA 2

Gestión de Información en la Web

Breve descripción

Desarrollo de un Sistema de Recomendación basado en Filtrado Colaborativo

Autor

Álvaro de la Flor Bonilla (alvdebon@correo.ugr.es) 15408846-L

Propiedad Intelectual

RESUMEN

En esta práctica se construirá, partiendo de cero y en el lenguaje preferido, un sistema de recomendación de películas basado en filtrado colaborativo (usuario - usuario). Para tal fin, se desarrollará una aplicación que muestre al usuario 20 películas al azar y éste las evalúe asignándole un valor de 1 estrella (*), indicando que no le gusta nada, hasta 5 estrellas (es una de sus películas favoritas).

Una vez obtenidas las valoraciones del usuario, el sistema procederá a calcular el vecindario, es decir, el grupo de usuarios que más se parecen a él en cuanto a las películas vistas y a las valoraciones dadas.

Finalmente, la aplicación predecirá la valoración para el usuario activo de todas las películas que han visto sus vecinos y que no ha visto el usuario activo, y le mostrará aquellas predichas con cuatro o cinco estrellas.

ÍNDICE DEL PROYECTO

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Colección

Para la realización de este proyecto se ha utilizado el conjunto de datos provisto por *MovieLens*, en concreto se ha usado el pequeño que aglutina 100 000 puntuaciones, 1682 películas y 943 usuarios.

1.1.1 IDE de desarrollo

Para el desarrollo de la práctica hemos utilizado PyCharm 2020.2.

2 DESCRIPCIÓN DEL DESARROLLO

Este proyecto se basa en dos módulos principales. El primer módulo es el encargado gestionar la base de datos y tratar con *"MongoDB Atlas"*. El segundo módulo es el encargado de realizar el sistema de recomendación.

2.1 Base de datos

Se han diseñado tres clases para configurar el esqueleto de nuestra aplicación.

- **Usuarios (USER)** que son las personas que realizan las valoraciones. Esta clase queda definida por los atributos *"user_id"*, *"age" (edad)*, *"gender"* (género), *"occupation"* (ocupación) y *"zip_code"* (código postal).

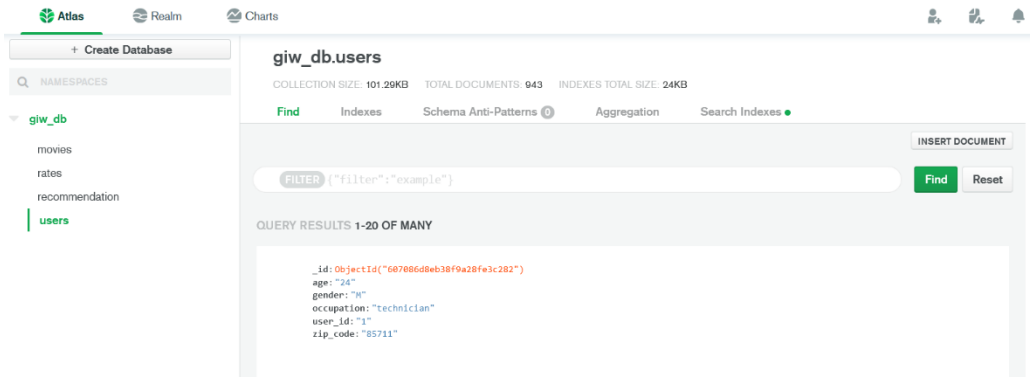

Ilustración 1 - Users

- **Películas (MOVIE)** que es la propia película que está siendo juzgada/valorada por el conjunto de los usuarios. Esta clase queda definida por los atributos *"movie_id"*, *"title"* (título), *"release_date"* (fecha de lanzamiento), *"video_release_date"*, *"imdb_url"* y *"genres"* (géneros).

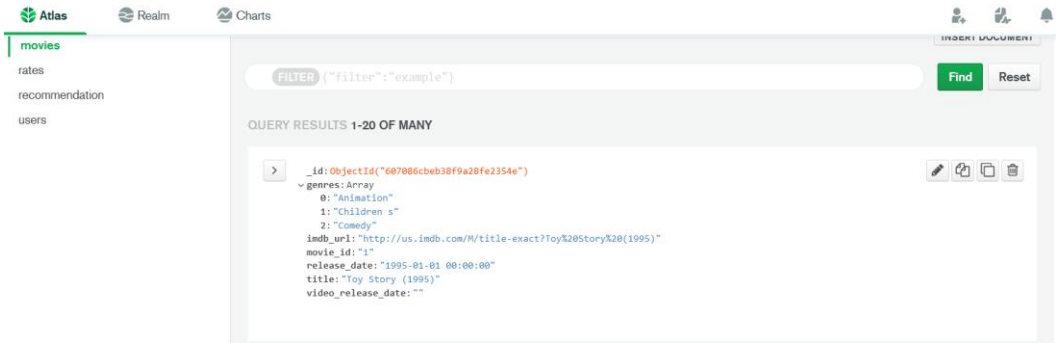

Ilustración 2 - Movies

- **Valoración (RATE)** que son las valoraciones realizadas por los usuarios sobre las distintas películas. Finalmente, esta clase la define el atributo *"user_id"* (identificador del usuario), *"item_id"* (identificador de la película), *"rating"* (puntuación) y *"timestamp"* (fecha de publicación).

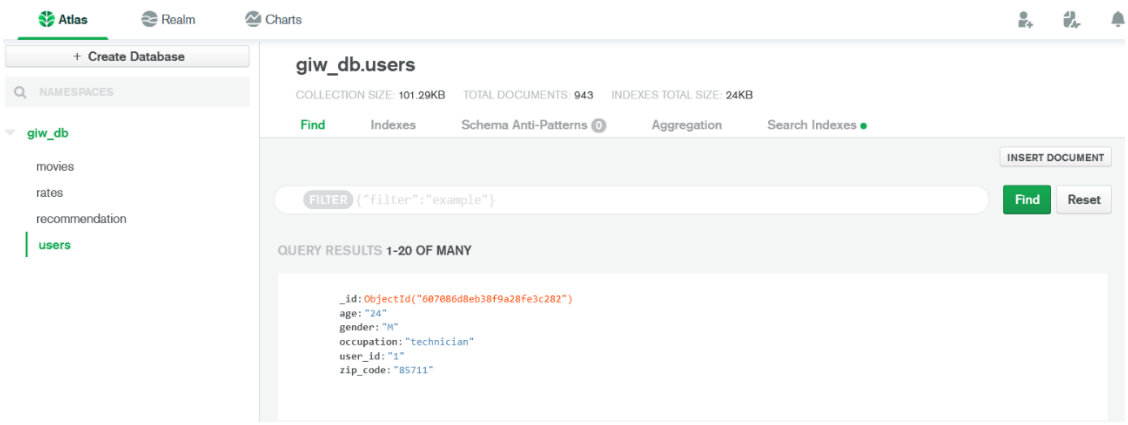

Ilustración 3 - Users

```python
def parse_date(date):
    new_date = ""
    if date:
        new_date = datetime.strptime(date, '%d-%b-%Y')
    return new_date


class Movie:
    def __init__(self, movie_id, title, release_date, video_release_date, imdb_url, genres):
        self.movie_id = movie_id
        self.title = title
        self.release_date = parse_date(release_date)
        self.video_release_date = parse_date(video_release_date)
        self.imdb_url = imdb_url
        self.genres = genres

    def to_json(self):
        self.release_date = self.release_date.__str__()
        self.video_release_date = self.video_release_date.__str__()
        return json.dumps(self, default=lambda o: o.__dict__,
                          sort_keys=True, indent=4)
```

Ilustración 4 - Esquema de clase principal

De manera general, las tres clases de objetos principales siguen el esquema que puede verse en la ilustración superior. Destacar el método para serializar en formato JSON cada una de las clases para facilitar el tratamiento con MongoDB.

Para cada una de estas clases se ha realizado un módulo auxiliar encargado de trabajar con la base de datos. Estas clases realizan la lectura del fichero sin tratar, construye un objeto por cada uno de los datos recogidos y se encargan de almacenarlos en la nube, en este caso haciendo uso de *"MongoDB Atlas"*.

De forma adicional, cuando finalmente se obtiene el resultado de recomendación que se le ofrece al usuario, también queda almacenada en la base de datos, pensando en un posterior procesamiento de estos datos.

De manera general, cada una de las clases principales cuentan por tanto con los siguientes métodos (***"*"*** varía en función de la clase principal utilizada):

- **read_*_*(url_data)**

Utiliza el fichero de *"MovieLens"*, trata los datos y crea un objeto por cada una de las líneas y devuelve el conjunto generado.

- **save_*(items)**

En este método son almacenados en *"MongoDB Atlas"* el conjunto de datos creados en el método anterior.

- **to_class(item)**

Deserializador para transformar el JSON recibido al realizar las búsquedas en MongoDB.

- **get_all_*()**

Obtener todos los objetos almacenados de una determinada clase.

- **filter_*(param_1, param_2, param_x…)**

  Localiza el conjunto de objetos que coincidan con parámetros que se señalan como atributos.

De igual forma, como se comentó anteriormente se ha creado el método **save_user_rate_search(rates, recommendations)**. Este método guarda en forma de tupla (en MongoDb), la valoración que el usuario realiza a veinte películas aleatorias y el conjunto de veinte películas que el sistema le recomienda a partir de estas valoraciones.

2.2 Recomendador

El primer paso que se ha realizado es la construcción de un diccionario estático (mediante el uso de la librería *"shelve"*). En este diccionario por cada usuario almacenaremos la valoración realizada a cada una de las películas y lo almacenamos de forma persistente en la carpeta indicada por el usuario en el archivo *"constant.py"* (de forma predeterminada (en *"./data/p2.dat.dat."*).

El siguiente paso, continuando con la explicación dada en clase es realizar el cálculo de vecinos cercanos. Para ello en nuestro caso hemos utilizado como función de similitud la correlación de *"Pearson"*.

```
common_item = {}
for item in dict_user[user_1]:
    if item in dict_user[user_2]:
        common_item[item] = 1

# Sum calculations
size = len(common_item)

# If it do not share any rating, we return 0
if size == 0:
    return 0

# Sums of all the preferences
sum_1 = sum([dict_user[user_1][movie] for movie in common_item])
sum_2 = sum([dict_user[user_2][movie] for movie in common_item])

# Sums of the squares
sum_sqrt_1 = sum([pow(dict_user[user_1][movie], 2) for movie in common_item])
sum_sqrt_2 = sum([pow(dict_user[user_2][movie], 2) for movie in common_item])

# Sum of the products
product_sum = sum([dict_user[user_1][movie] * dict_user[user_2][movie] for movie in common_item])

# Calculate r (Pearson score)
numerator = product_sum - (sum_1 * sum_2 / size)
denominator = sqrt((sum_sqrt_1 - pow(sum_1, 2) / size) * (sum_sqrt_2 - pow(sum_2, 2) / size))

if denominator == 0:
    return 0

res = numerator / denominator
return res
```

Ilustración 5 - Cálculo de correlación de Pearson

Una vez calculada la correlación con los distintos vecinos utilizaremos esta matriz de resultados para ponderar las diferentes películas no vistas. En concreto se realiza en el método *"get_recommendation()"*.

```
weighted_score = {}
similarity_sums = {}
for other in dict_user:
    # Don't compare the same user
    if other == person:
        continue
    similarity_result = similarity(dict_user, person, other)
    # Ignore scores of zero or lower
    if similarity_result <= 0:
        continue
    for movie in dict_user[other]:
        # We add movies not seen by the user
        if movie not in dict_user[person] or dict_user[person][movie] == 0:
            # Similarity * Score
            weighted_score.setdefault(movie, 0)
            weighted_score[movie] += dict_user[other][movie] * similarity_result
            # Sum of similarities
            similarity_sums.setdefault(movie, 0)
            similarity_sums[movie] += similarity_result

# Create the normalized list
rankings_aux = [(total / similarity_sums[item], item) for item, total in weighted_score.items()]
```

Ilustración 6 - Puntuar películas no vistas

3 MANUAL DE USUARIO

Al ejecutar nuestro programa aparecerá un menú similar a la siguiente imagen.

```
1   Ejecutar recomendador
2   Volver a cargar base de datos
0   Salir

Desición (opciones: ['1', '2', '0']):
```

Ilustración 7 - Menu principal

La opción 2 vuelva a realizar la lectura de todos los archivos y la carga en la base de datos de MongoDB Atlas.

Por otro lado, la opción 1 ejecuta el recomendador en sí que se requiere en la práctica.

```
1   Ejecutar recomendador
2   Volver a cargar base de datos
0   Salir

Desición (opciones: ['1', '2', '0']): 1
Establezca su valoración para las siguientes películas 20 películas


1   Braveheart (1995)
Inserte su valoración
Desición (opciones: ['1', '2', '3', '4', '5']): 7
Opción, no válida, inténtelo de nuevo. Intento 1 (máximo: 3)
Inserte su valoración
Desición (opciones: ['1', '2', '3', '4', '5']):
```

Ilustración 8 - Ejecución del recomendador

Como primer paso para recomendar 20 películas al usuario se le solicitará que califique 20 películas seleccionadas aleatoriamente en cada ejecución. Para cada una de estas películas se solicita que se inserte una valoración de 1 (peor) a 5 (mejor). Se le concede al usuario 3 intentos para insertar un valor válido, en caso contrario se estable la puntuación tres como valor por defecto.

Cuando se introducen los 20 valores de las películas aleatorias comienza el proceso para localizar la película recomendada. Una vez finalizado, se muestran las 20 seguidas, ordenadas de más o menos calificación (según la matriz calculada y ponderación obtenida) y queda almacenado en base de datos los resultados obtenidos.

Ilustración 9 - Recommendation