Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

ESCUELA POLITÉCNICA.

GRADO EN INGENIERÍA DE INFORMÁTICA.

Recomendaciones MovieLens

Sistemas Inteligentes y Representación del Conocimiento

Javier Relinque Rodríguez

Pablo Ferrer López

Índice

Descripción de la base de datos 3
Descripción de las tablas y sus atributos 3
Datos de conexión con la BD 3
Scripts de la creación de las tablas 3
Script de la carga de datos 4
Problemas y dificultades 6
Recomendaciones User-User 6
Código 6
User_similitude 7
Prediction 8
Ranking 9
JavaScript 10
Casos de Uso 11
Ránking 11
Problemas y dificultades 14
Recomendaciones Item-Item 15
Item_similitude 15
Prediction 17
Ranking 18
Problemas y dificultades 19
Gestión de un nuevo usuario 20
Casos de Uso 20
Descripción del portal web 22
Manual de instalación 22
Bibliografía 23

Descripción de la base de datos

Para esta práctica utilizaremos la información almacenada en archivos tipo .csv provistos por la página web “Movie Lens” siendo estos almacenados en una base de datos del tipo MySQL.

Descripción de las tablas y sus atributos

Tenemos dos archivos, uno de nombre “movies” el cual contiene el id de las películas, su título y sus géneros. En este caso para nuestra base de datos nos hemos quedado con el id de las películas tratándolo como primary key auto_increment de tipo int con tamaño 11; también hemos utilizado los títulos de las películas como tipo varchar con tamaño 255 el cual utiliza el “encoding” utf-8.

El segundo archivo, de nombre “ratings” contiene los ids de los usuarios que hayan puntuado una película, el id de la película correspondiente, la puntuación otorgada a esa película y finalmente una variable de tiempo. Para nuestra base de datos hemos utilizado el id de los usuarios como primary key de tipo int con tamaño 11, el id de las películas como foreign key de tipo int con tamaño 11 y el rating de cada película como tipo float.

Datos de conexión con la BD

Dado que el proyecto se maneja de manera local, decidimos utilizar los valores estándares para el “host”, “user” y “password”, siendo estos “localhost”, “root” y de contraseña nula.

En el momento en que este proyecto o cualquier otro deje de ser local estos datos serían modificados y se asignaría una contraseña.

Scripts de la creación de las tablas

Para generar este script hemos decidido utilizar Python como lenguaje dada su versatilidad.

El nombre de este archivo con extensión “.py” es “**DBScript**” y a continuación explicaremos paso a paso su funcionamiento:

Inicialmente, importamos el conector de la base de datos MySQL y asignamos los datos de conexión. Después creamos un cursor con esos datos y a través de este ejecutamos la “query” que crea la base de datos. Tras esto cerramos el cursor.

```
import mysql.connector

class DBScript():
    db = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
    )
    cursor = db.cursor()
    cursor.execute("CREATE DATABASE IF NOT EXISTS Sistemas_de_recomendacion")

    cursor.close()
```

Después de esto volvemos a establecer conexión con los datos anteriormente mencionados solo que esta vez conectamos con la base de datos ya creada:

```
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="sistemas_de_recomendacion",
    use_unicode=True,
    charset="utf8",

)
```

Una vez establecida la conexión, volvemos a crear un cursor y primero ejecutaremos una "query" para poder asignar a nuestra base de datos el encoding utf-8.

Una vez asignado, creamos ambas tablas y después cerramos el cursor:

```
cursor2 = db.cursor()
cursor2.execute("ALTER DATABASE Sistemas_de_recomendacion CHARACTER SET"
                " 'utf8' COLLATE 'utf8_unicode_ci'")

cursor2.execute("CREATE TABLE IF NOT EXISTS movies (id INT AUTO_INCREMENT PRIMARY KEY,"
               " title VARCHAR(255))")

cursor2.execute("CREATE TABLE IF NOT EXISTS ratings (userid INT, movieid INT, "
                "rating FLOAT, FOREIGN KEY(movieid) REFERENCES movies(id))")

cursor2.close()
```

Script de la carga de datos

Una vez tenemos ya creada la base de datos procederemos a realizar la carga de los archivos de extensión .csv.

Para ello hemos decidido utilizar la librería "pandas".

Inicialmente establecemos conexión con nuestra base de datos y declararemos una serie de variables que utilizaremos para almacenar la información de los archivos:

```
import mysql.connector
import pandas as pd


class CSVScript():

    db = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="sistemas_de_recomendacion",
        charset="utf8mb4",
        use_unicode=True

    )

    titles = []
    userid = []
    movieid = []
    rating = []
```

Después utilizando la librería pandas y un sistema de variables booleanas (semáforos) almacenaremos la información en nuestras variables:

```
    bool=True
    bool1=True
    bool2=True

    data = pd.read_csv("./movies.csv")
    data = data.values.tolist()

    for a in data:
        for b in a:
            if bool == True:
                bool=False
                bool1=False
            elif bool1 == False:
                titles.append(str(b))
                bool1 = True
                bool2 = False
            elif bool2 == False:
                bool=True
                bool1=True


bool=True
bool1=True
bool2=True
bool3=True

data2 = pd.read_csv("./ratings.csv")
data2 = data2.values.tolist()

for i in data2:
    for j in i:
        if bool == True:
            bool = False
            bool1 = False
            userid.append(int(j))
        elif bool1 == False:
            bool1 = True
            bool2 = False
            movieid.append(int(j))
        elif bool2 == False:
            bool2 = True
            bool3 = False
            rating.append(float(j))

        elif bool3 == False:
            bool3 = True
            bool = True
```

Finalmente abriremos un cursor de la base de datos y recorreremos nuestras listas insertando la información en esta:

```
cursor = db.cursor()

i = 0

for i in range(0, len(titles)):
    query = """INSERT INTO movies (title) VALUES (%s)"""
    cursor.execute(query, [titles[i]])
    i = i + 1

j = 0
for j in range(0, len(userid)):
    query2 = """INSERT INTO ratings (userid,movieid,rating) VALUES (%s,%s,%s)"""
    cursor.execute(query2, [userid[j],movieid[j],rating[j]])
    j = j + 1
cursor.close()

db.commit()
```

Problemas y dificultades

Debido a causas desconocidas nos fue imposible asignar el "encoding" utf-8 directamente en la extracción de los datos de los archivos de extensión .csv o en la asignación de los datos de la base de datos por lo que nos fue necesario realizar una "query" ALTER DATABASE en la que se le asignó.

Otro problema fue que a la hora de utilizar la librería "pandas" no pudimos encontrar las funciones que permitían acceder directamente a cada columna y he ahí el porqué de que utilizásemos la estructura de semáforos.

Recomendaciones User-User

El primer apartado de la página web es el referente a las recomendaciones User-User, lo que significa que usamos el algoritmo de Pearson de similitud entre usuarios

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\ \sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}$$

Con este coeficiente, podemos llegar a sacar el valor predicho para una película dada, asumiendo que haya sido votada por alguno de los vecinos de nuestro usuario dado.

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

Código

El fichero "db.php" se ocupa de la conexión a base de datos, y desde "index.php" llamamos a todos los demás.

"useruser.php" contiene el código html necesario para mostrar las dos opciones que tenemos en este apartado, siendo la de mostrar el ranking de películas y el de predecir el valor de una película determinada. Para mostrar todos los usuarios y todas las películas, hacemos llamadas a nuestras funciones de php y las rellenamos con un bucle.

```
<form>
    <h3>Predecir puntuación para una película</h3>
    <h4>Selecciona un usuario:</h4>
    <select name="selectuser2" id="selectuser2">
        <?php foreach ($user_list as $user){
            echo "<option value=$user>ID $user</option>";
        }?>
    </select>

    <h4>Selecciona una película:</h4>

    <select name="selectmovie" id="selectmovie">
        <?php foreach ($movie_list as $movie){
            echo "<option value=$movie[0]>$movie[0]: $movie[1]</option>";
        }?>
    </select>
    <h4>Umbral de similitud: <input type="text" id="threshold2" value="0.75"></input></h4>
    <input type="button" id="predict" value="¡Predecir!"></input>
</form>
```

Todas las funciones que utilizamos en este apartado se llaman desde el fichero "userfunctions.php". Los dos primeros métodos, "user_list" y "movie_list", simplemente incluyen una llamada a la base de datos para devolvernos todos los usuarios y todas las películas.

User_similitude

Nuestra siguiente función es "user_similitude", que se ocupa de calcular todas las similitudes user-user para un usuario dado. Con este array, podemos realizar la predicción de cualquier película, así que vamos a explicar paso a paso esta función.

```
$sql_query = "SELECT DISTINCT ratings1.movieid, ratings1.rating AS rating1, ratings2.rating AS rating2 FROM ratings AS ratings1,
ratings AS ratings2 WHERE ratings1.userid = '$userid' AND ratings2.userid='$user' AND ratings1.movieid = ratings2.movieid";

$result = $con->query($sql_query);
if (mysqli_num_rows($result)!=0) {

$movielist2 = array();
$ratings1 = array();
$ratings2 = array();
while($row = $result->fetch_assoc()){

    array_push($movielist2, array($row['movieid'], $row['rating1'], $row['rating2']));
    array_push($ratings1, $row['rating1']);
    array_push($ratings2, $row['rating2']);
}
```

En primer lugar, para cada usuario($user), necesitaremos saber qué películas tiene en común con nuestro usuario argumento ($userid), y la puntuación que ambos han dado. Guardamos todos esos valores en un único array, y como la media de puntuaciones que se usa a la hora de calcular la similitud es sólo la media de puntuaciones de las películas que tienen en común los dos usuarios, guardamos también todas las puntuaciones en dos arrays distintos ($ratings1 siendo el de nuestro usuario argumento, $ratings2 el del usuario con el que lo comparaos).

```php
$average1 = array_sum($ratings1)/count($ratings1);
$average2 = array_sum($ratings2)/count($ratings2);

$numerador = 0;
$denominador1 = 0;
$denominador2 = 0;
foreach($movielist2 as $movie){
    $numerador += ($movie[1] - $average1)*($movie[2] - $average2);
    $denominador1 += pow(($movie[1] - $average1),2);
    $denominador2 += pow(($movie[2] - $average2),2);
}

if($numerador == 0){
    $similitude = $numerador;
}else{
    $similitude = $numerador/(sqrt($denominador1)*sqrt($denominador2));
}
array_push($similitudelist,array($user,$similitude));

}
```

Con esos dos arrays, resulta simple calcular la media. Después, empezamos a aplicar la fórmula de la similitud de Pearson. Por ello, debemos restar a la puntuación que cada uno de nuestros dos usuarios ha puesto, la media de sus puntuaciones. Incrementamos nuestro numerador y las dos partes independientes del denominador siguiendo la lógica de los sumatorios definidos por Pearson, y tan solo al final los dividimos. Antes de ello, no obstante, comprobamos que el numerador no sea 0, puesto que si lo es, también lo será el denominador, y si intentamos aplicar la fórmula de Pearson, nos dará un fallo al intentar dividir entre cero.

Si hemos conseguido aplicar la fórmula de Pearson correctamente, entonces guardamos en un array el usuario comparado junto a su coeficiente de correlación.

Prediction

Con todas las similitudes dadas, tenemos suficiente como para poder calcular la puntuación de una película no puntuada por nuestro usuario.

```php
if($similitude == 'null'){
    $similitude = user_similitude($userid);
}
$trulysimilarusers = array();
$movierating = array();

foreach($similitude as $user){
    if($user[1] >= $umbral and $user[0] != $userid){
        array_push($trulysimilarusers, $user);
    }
}

foreach($trulysimilarusers as $user){

    $sql_query = "SELECT rating FROM ratings WHERE movieid='$movieid' AND userid='$user[0]'";
    $result = $con->query($sql_query);
    if (mysqli_num_rows($result)!=0) {
        while($row = $result->fetch_assoc()){
            array_push($movierating, array($user[0], $movieid, $user[1], $row['rating']));
        }
    }
}

if(empty($movierating)){
    return "No se puede calcular para los valores dados";
}
```

Comenzamos con una serie de comprobaciones. Es una decisión consciente la de permitir calcular la predicción de una película que el usuario sí haya votado, por si se desea comparar el valor predicho con el real.

En primer lugar, se comprueba si se ha recibido el array de similitudes como argumento y, si no, se calcula. Esta comprobación se hace porque en el ranking calcularemos la similitud antes de empezar a calcular predicciones (para ahorrar tiempo de ejecución), pero en el caso de calcular la predicción de una película dada llamaremos directamente a este método.

Después, reducimos el número de usuarios que vamos a usar para nuestro cálculo (es decir, nuestro vecindario) según el umbral dado como argumento. Como el propio usuario siempre tendrá un coeficiente de similitud de 1, también lo excluimos de la lista.

A continuación, buscaremos la puntuación dada a la película por parte de cada usuario de nuestro vecindario. Nuestro nuevo array tendrá los valores de similitud, id del usuario, y puntuación para la película dada. Si una vez concluimos nuestra búsqueda este nuevo array ($movierating) está vacío, significa que ninguno de los usuarios del vecindario ha puntuado la película, y por lo tanto no podemos calcular una predicción.

```
$sql_query = "SELECT AVG(rating) AS rating FROM ratings WHERE userid='$userid'";
$result = $con->query($sql_query);
while($row = $result->fetch_assoc()){
    $average = $row['rating'];
}

$numerador = 0;
$denominador = 0;

foreach($movierating as $score){
    $sql_query = "SELECT AVG(rating) AS rating FROM ratings WHERE userid='$score[0]'";
    $result = $con->query($sql_query);
    while($row = $result->fetch_assoc()){
        $average2 = $row['rating'];
    }

    $numerador += ($score[2] * ($score[3] - $average2));
    $denominador += ($score[2]);

}

$prediccion = $average + $numerador/$denominador;
return $prediccion;
```

Como las medias usadas para la predicción son las medias globales de los usuarios, podemos obtener el valor directamente de la base de datos. Después simplemente calculamos el valor con la fórmula de Pearson.

Ranking

El ranking de películas es un método simple, en el que calculamos la predicción de todas las películas y luego las ordenamos. No obstante, hay una serie de comprobaciones que merecen la pena ser discutidos.

```
$similitude = user_similitude($userid);
$trulysimilarusers = array();
foreach($similitude as $user){
    if($user[1] >= $umbral and $user[0] != $userid){
        array_push($trulysimilarusers, $user);
    }
}

foreach($trulysimilarusers as $similar){
    $sql_query = "SELECT movieid FROM ratings WHERE userid = '$similar[0]' AND movieid NOT IN  (SELECT movieid FROM ratings WHERE userid = '$userid')";
    $result = $con->query($sql_query);
    while($row = $result->fetch_assoc()){
        if (in_array($row['movieid'], $movielist) == false){
            array_push($movielist, $row['movieid']);
        }
    }
}
```

En primer lugar, calculamos el array de similitudes de nuestro usuario dado para evitar tener que calcularlo por cada predicción, y después, seleccionamos los usuarios por encima del umbral del mismo modo que hicimos a la hora de calcular la predicción.

Para reducir el número de películas para las que calcularemos la predicción, escogemos tan solo las películas que han sido puntuadas por los usuarios del vecindario, pero no han sido puntuadas por nuestro usuario. Con este array, ya podemos iterar para cada película y calcular la predicción.

```
$order = array_column($predictlist, 1);
array_multisort($order, SORT_DESC, $predictlist);
array_splice($predictlist, $limite);

$resultlist = array();

foreach($predictlist as $predict){
    $sql_query = "SELECT title FROM movies WHERE id = '$predict[0]'";
    $result = $con->query($sql_query);
    if (mysqli_num_rows($result)!=0) {
        while($row = $result->fetch_assoc()){
            array_push($resultlist, array($predict[0], $row['title'], $predict[1]));
        }
    }else{
        array_push($resultlist, array($predict[0], "Título no disponible", $predict[1]));
    }

}
```

Con esto, ordenamos nuestro array resultado con array multisort. No teníamos otro modo más sencillo al estar trabajando continuamente con arrays multidimensionales, y una vez lo hemos ordenado, recortamos nuestro array para mostrar tan solo el número de películas que se ha indicado (por defecto, 5).

Después, seleccionamos nuestro array final y buscamos para cada película del ranking el título de la película. Como usamos el dataset pequeño, con tan solo unas 9000 películas en la tabla de movies pero más de 100000 en la de ratings, existe la posibilidad de que predigamos películas que no tienen un título en nuestra base de datos, por lo que en esos casos lo indicamos con un “Título no disponible”.

JavaScript

A las funciones descritas en “userfunctions.php” las llamaremos desde “useruser.js”. Como la página web es dinámica, utilizamos jQuery y Ajax para obtener los resultados seleccionados o escritos, y llamamos a los correspondientes scripts: “ranking.php” y “prediction.php”

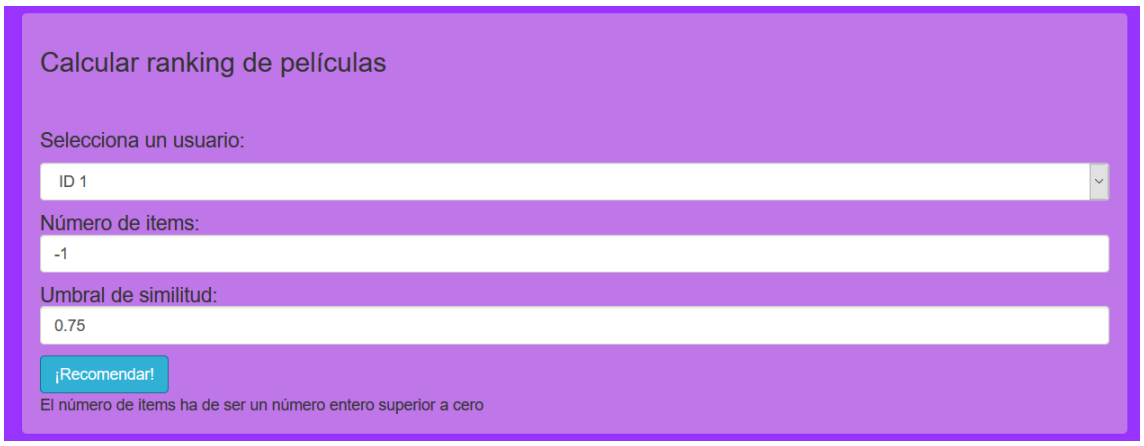
```php
if ((isset($_REQUEST['userid'])) and (isset($_REQUEST['umbral']))and (isset($_REQUEST['limite']))){
    if((! ctype_digit(strval($_REQUEST['limite']))) or ($_REQUEST['limite'] === '0')){
        echo "El número de items ha de ser un número entero superior a cero";
    }else if((!is_numeric($_REQUEST['umbral']))){
        echo "El umbral ha de ser un número";
    }else{

        $ranking = ranking($_REQUEST['userid'], $_REQUEST['umbral'], $_REQUEST['limite']);
        echo "
                <table>
                    <tr>
                        <th>Película</th>
                        <th>Puntuacion Predicha</th>
                    </tr>
        ";
        foreach ($ranking as $rank){
            echo "
                    <tr>
                        <th>",$rank[0],": ",$rank[1],"</th>
                        <th>",$rank[2],"</th>
                    </tr>";
            }


            echo"

                </table>
            ";

    }
}else{
    echo "Escriba todos los datos";
}
```

Nuestro script comprueba tanto que hemos dado valores para todos los argumentos, como que esos valores son válidos. El umbral puede ser superior a 1 e inferior a -1, porque no fue posible incluir esa comprobación en el script debido al modo que $_REQUEST lee los datos, pero aun así funciona de forma coherente. Es decir, si se escoge un umbral por encima de 1 no se podrá calcular ninguna película, y por debajo de -1 cogerá a todos los usuarios.

Casos de Uso

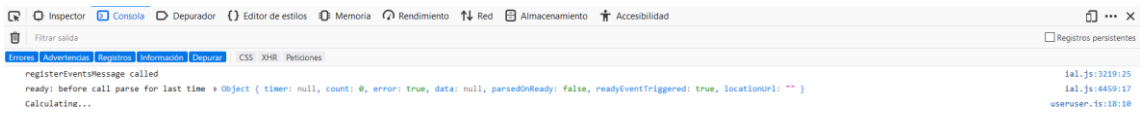
Ranking

Por defecto, estos son los valores indicados para el ranking. Como control de errores, el programa nos indicará si hemos introducido algún valor erróneo.

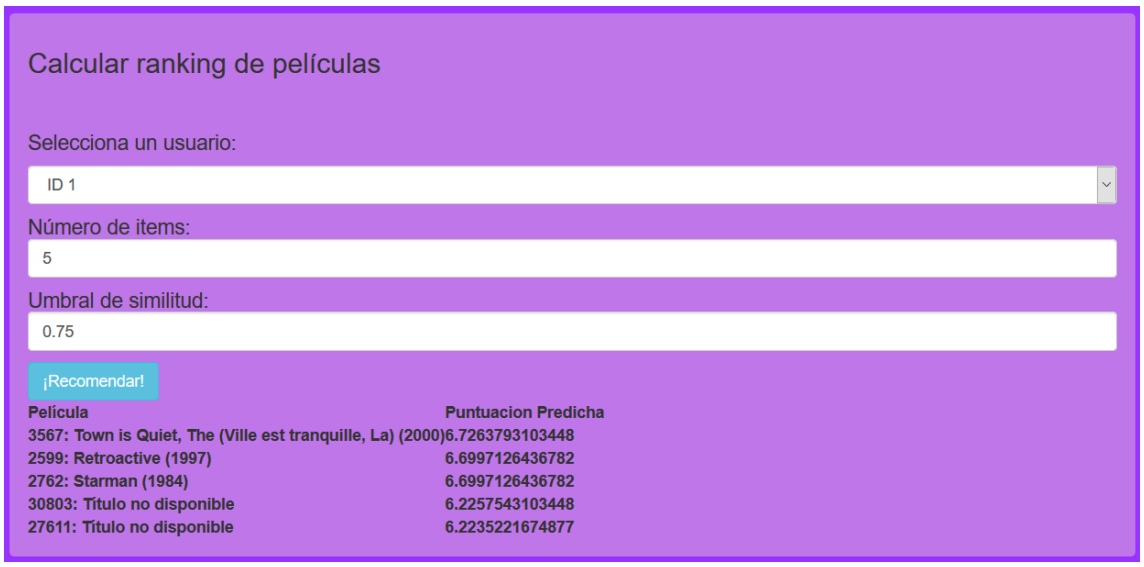


Asumiendo que todo sea correcto, nos mostrará el siguiente mensaje.

Como las operaciones pueden llevar bastante tiempo, una forma adicional de asegurarse de que la página está calculando es comprobar la consola desde las herramientas para desarrolladores.

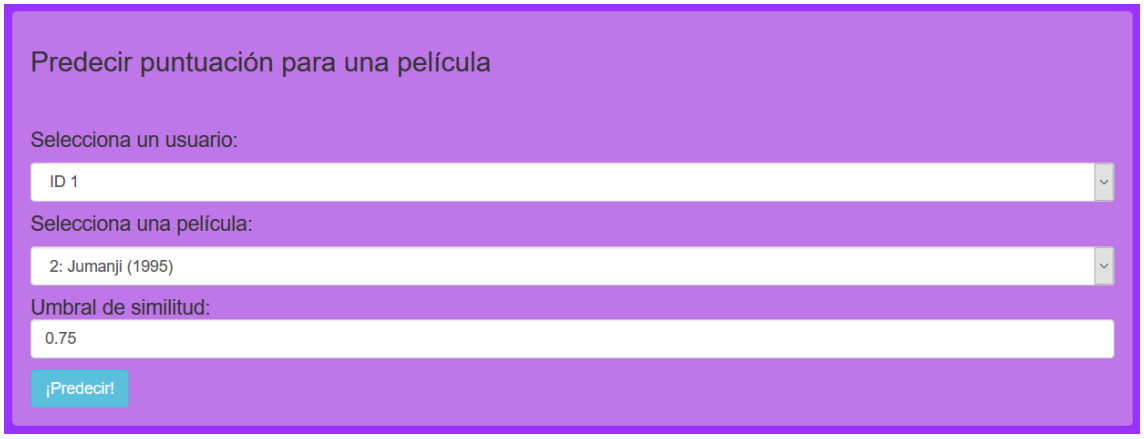


Si el último mensaje es “Calculating…”, es que la función no ha dado un fallo y aún está calculando.

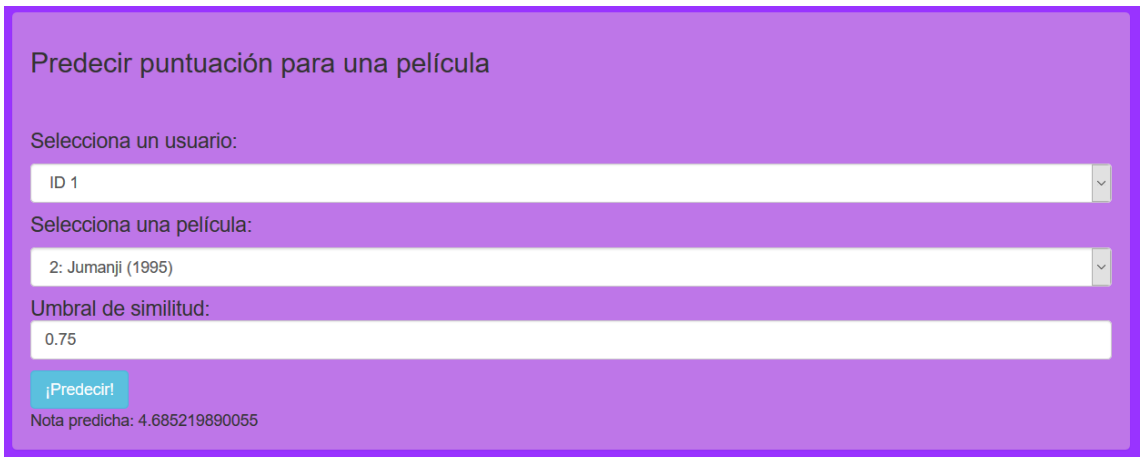


Si todo sale bien, nos mostrará nuestras películas predichas. Las películas cuyo id sea superior a 9742 no tienen título al no aparecer en la tabla de "movies".

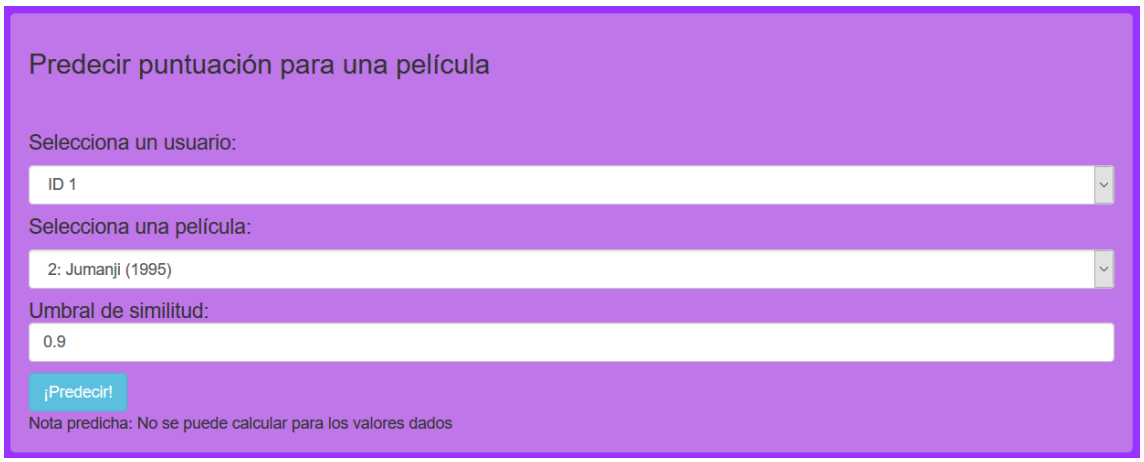
Predicción



Para predicciones, tenemos el mismo tipo de control de errores que en el ranking.



Si todo funciona, nos mostrará la puntuación que predice para nuestra película.



Si nuestro umbral es demasiado exigente, no podremos crear un vecindario con el que puntuar la película, por lo que nos dará este mensaje de error.

Problemas y dificultades

El mayor problema encontrado es el del tiempo de ejecución, que es muy grande. Como se usan múltiples bucles, llamadas a base de datos, y arrays multidimensionales, el tiempo de ejecución aumenta exponencialmente cuanto más pequeño sea el umbral.

Para las pruebas, XAMPP permite un tiempo máximo de ejecución de un minuto, que tuvo que incrementarse editando el fichero “php.ini” y aumentando el tiempo hasta más de media hora (aunque no llega a gastarse, con el ranking necesitando en torno a 15 minutos, a lo sumo). Con WAMPP el tiempo de ejecución es mucho más reducido, pero no se consiguió aumentar la velocidad con XAMPP ni usando todas las soluciones propuestas en las primeras páginas de sugerencias de Google, por lo que se acabó tomando el tiempo de ejecución como aceptable.

A la hora de trabajar, se vio necesario el usar arrays multidimensionales para guardar todos los valores necesarios para el cálculo tanto del coeficiente de similitud como la nota predicha. Esto no solo hizo que la depuración de código fuese más compleja, sino que también forzó a usar métodos como array_multisort.

Para la comprobación de la veracidad de los valores dados, se usaron calculadoras online de similitudes de Pearson, usando como ejemplo usuarios con solo unas pocas películas en común (como el usuario 1 y el usuario 3).

La comprobación de la predicción llevó a más problemas, puesto que incluso con películas de cálculos sencillos, nuestro valor predicho era superior a 5 (supuestamente, el máximo). Incluso revisando el cálculo y simulándolo con los Excels proporcionados por la asignatura, estos valores no variaban. Tanto en nuestros apuntes como en varias páginas web se menciona cómo el cálculo de predicciones de Pearson no es realmente fiable para bases de datos “pequeñas” (como la nuestra), o con vecindarios pequeños (como los que nosotros usamos normalmente). Por eso hemos mantenido los resultados por encima de 5 sin edulcorar. Podríamos haberlos editado para que los valores predichos por encima de 5 no contaran o se tomasen como 5, pero hemos decidido mantenerlos con sus valores calculados realmente no solo para que sea más fácil la comprobación de que se han aplicado los algoritmos correctamente (si fuera necesario), sino también para que se muestre el ranking genuino.

Recomendaciones Item-Item

El segundo apartado de la página web es el referente a las recomendaciones Item-Item, lo que significa que usamos el algoritmo de Pearson de similitud entre películas.

$$sim(a,b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}}$$

Dado que trabajamos con películas es necesario obtener la similitud de cada película con el resto para poder obtener un valor de predicción.

$$pred(u,p) = \frac{\sum_{i \in ratedItems(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ratedItems(u)} sim(i,p)}$$

"itemitem.php" contiene el código html necesario para mostrar las dos opciones que tenemos en este apartado, siendo la de mostrar el ranking de películas y el de predecir el valor de una película determinada.

Para mostrar todos los usuarios y todas las películas, hacemos llamadas a nuestras funciones de php y las rellenamos con un bucle.

```
<form>
    <h3>Predecir puntuación para una película</h3>
    <h4>Selecciona un usuario:</h4>
    <select name="selectuser2" id="selectuser2">
        <?php foreach ($user_list as $user){
            echo "<option value=$user>ID $user</option>";
        }?>
    </select>

    <h4>Selecciona una película:</h4>

    <select name="selectmovie" id="selectmovie">
        <?php foreach ($movie_list as $movie){
            echo "<option value=$movie[0]>$movie[0]: $movie[1]</option>";
        }?>
    </select>
    <h4>Umbral de similitud: <input type="text" id="threshold2" value="0.75"></input></h4>
    <input type="button" id="predict" value="¡Predecir!"></input>
</form>
```

Todas las funciones que utilizamos en este apartado se llaman desde el fichero "itemfunctions.php". Los dos primeros métodos, "user_list" y "movie_list", simplemente incluyen una llamada a la base de datos para devolvernos todos los usuarios y todas las películas.

Item_similitude

Nuestra siguiente función servirá para poder calcular las similitudes de cada una de las películas entre sí para poder aplicar la función de similitud item-item.

Esta función recibirá como parámetros la película y el usuario para el cual queremos predecir una puntuación.

Inicialmente para generar la lista de películas que querremos recorrer para ir sacando las similitudes, hemos realizado una "query" en la cual, para poder optimizar el proceso, hemos obtenido aquellas películas las cuales cumplan los requisitos de que sus usuarios hayan puntuado la película que queremos predecir y hayan puntuado también una o mas películas que ha puntuado el usuario que recibe la función. Tras esto, almacenamos los resultados en una variable "array" de nombre $movielist y declaramos un par de variables "array" que utilizaremos posteriormente.

```
function item_similitude($movieid, $userid){
    include "db.php";
    $movielist = array();
    $query = "SELECT DISTINCT rating2.movieid as movieid FROM ratings as rating1, ratings as rating2
    where rating1.movieid='$movieid' and rating1.userid = rating2.userid ORDER BY `rating2`.`movieid` ASC LIMIT 50";
    $result = $con->query($query);
    if (mysqli_num_rows($result)!=0) {
            while($row = $result->fetch_assoc()){
            array_push($movielist, $row['movieid']);
            }
        }
    $similitudelist = array();
    $ratinglist = array();
```

Una vez realizado esto procederemos a recorrer dicha lista de películas y por cada una de ellas realizaremos una "query" en la que la compararemos con nuestra película original. Es decir, para cada película obtendremos qué usuario la han votado y qué puntuación le han dado tanto a ésta como a nuestra película original.

Una vez tenemos estos datos los almacenamos en una serie de variables "array" para su posterior uso.

```
foreach($movielist as $movie){
    $sql_query = "SELECT DISTINCT ratings1.userid, ratings1.rating AS rating1, ratings2.rating AS rating2 FROM ratings
    AS ratings1, ratings AS ratings2 WHERE ratings1.movieid = '$movieid' AND ratings2.movieid='$movie' AND
    ratings1.userid = ratings2.userid";

    $result = $con->query($sql_query);

    if (mysqli_num_rows($result)!=0) {
        $ratings1 = array();
        $ratings2 = array();

        while($row = $result->fetch_assoc()){
            array_push($ratinglist, array($row['userid'], $row['rating1'], $row['rating2']));
            array_push($ratings1, $row['rating1']);
            array_push($ratings2, $row['rating2']);
        }
    }
```

Ahora que ya tenemos los datos suficientes podemos empezar a realizar los cálculos necesarios para obtener las similitudes de cada película. Para ello necesitaremos declarar una serie de variables donde almacenar los resultados y recorreremos la lista de puntuaciones obtenida anteriormente. Finalmente, este sería el resultado del cálculo de cada similitud:

```
    $numerador = 0;
    $denominador1 = 0;
    $denominador2 = 0;
    foreach($ratinglist as $ra){
        $sql_query2 = "SELECT AVG(rating) AS rat FROM `ratings` WHERE userid = '$ra[0]'";
        $result2 = $con->query($sql_query2);
        if (mysqli_num_rows($result2)!=0){
            while($row = $result2->fetch_assoc()){
                $average = $row['rat'];
            }
        }

        $numerador+=($ra[1] - $average)*($ra[2] - $average);
        $denominador1+=pow($ra[1] - $average,2);
        $denominador2+=pow($ra[2] - $average,2);
    }

    if($numerador!=0){
        $similitude = $numerador/(sqrt($denominador1)*sqrt($denominador2));
```

Será necesario obtener la puntuación de cada película en cada una de las iteraciones de nuestra lista de películas para la función que calcula la predicción.

```
    $query3 = "SELECT rating FROM ratings WHERE movieid = $movie AND userid = $userid";
    $result3 = $con->query($query3);
    if (mysqli_num_rows($result3)!=0){
            while($row = $result3->fetch_assoc()){
                $rating = $row['rating'];
                array_push($similitudelist, array($movie,$similitude,$rating));
            }
        }
    }
}
return $similitudelist;
```

Con esto último devolveremos una variable “array” multidimensional en la cual tendremos por cada posición la película su similitud y su puntuación.

Prediction

En nuestra función de predicción recibiremos los parámetros correspondientes a la película a predecir, el usuario para dicha película y el umbral de predicción.

Para esta función almacenaremos en una variable los resultados de llamar a la anterior función y comenzaremos a recorrer esta variable y calcularemos la predicción de dicha película.

```php
function prediction($userid,$movieid,$umbral){
    include "db.php";
    $result = array();
    $prediction = 0;
    $similitudes = item_similitude($movieid,$userid);
    if(empty($similitudes)){
        return "No se puede calcular para los valores dados";
    }
    $numerador = 0;
    $denominador = 0;
    foreach($similitudes as $sim){
        if($sim[1]>=$umbral){
            $numerador+=($sim[1])*($sim[2]);
            $denominador+=($sim[1]);
        }
    }
    if($numerador == 0 or $denominador == 0){
        return "No se puede calcular para los valores dados";
    }else{
        $prediction = $numerador/$denominador;
    }


    return $prediction;
    }
```

Ranking

La última función trata de obtener una lista de películas en función de los parámetros que le demos.

Esta función recibirá como parámetros el usuario para el cual queremos calcular el ranking, el umbral de similitud y un número tope de películas a mostrar.

Para esta función necesitaremos declarar un par de variables “array” en las cuales almacenaremos datos para su posterior uso y una variable que usaremos como contador.

```php
function ranking($userid,$umbral,$limit){
    include "db.php";
    $notseenmovies = array();
    $predictions = array();
    $counter = 0;
```

Inicialmente realizaremos una “query” en la cual obtendremos aquellas películas que no ha puntuado el usuario en cuestión.

```php
$userquery = "SELECT DISTINCT movieid FROM ratings WHERE userid != '$userid' AND movieid NOT IN
            (SELECT movieid FROM ratings WHERE userid = '$userid') ORDER BY movieid ASC LIMIT 100";
$result = $con->query($userquery);
while($row = $result->fetch_assoc()){
    $movie = $row['movieid'];
    array_push($notseenmovies, $movie);
}
```

Después recorreremos esta variable calculando para cada una de las películas su predicción y comprobando si ha sido posible realizar dicha predicción. En caso de recibir un "string" significará que la predicción para esa película no ha sido posible. Si no se recibe un string entonces se comprobará si la predicción es igual o superior a 5 (dado que como comentamos en user user es posible que las predicciones sean mayores que 5) y si lo es se asignará el valor "5" a la predicción para evitar errores y se sumará 1 a una variable contador. Por cada iteración de nuestra lista de películas no vistas por el usuario se comprobará si el contador coincide con nuestro tope de películas para el ranking y si lo es se devolverá la lista con las primeras películas encontradas. Si no llegase nunca a coincidir esto quiere decir que no ha conseguido encontrar ese número tope de películas con puntuación 5 y por lo tanto ordenará la lista y mostrará tantas películas como tope tengamos puesto.

Problemas y dificultades

El principal problema de este apartado es el gran volumen de información que hay que manejar ya que a diferencia de user-user en item-item debemos calcular las similitudes por cada una de las películas recorriendo la lista de películas por cada una de las películas en

```
    foreach($notseenmovies as $mov){
        if($counter == $limit){
            return $predictions;
        }
        $pred = prediction($userid,$mov,$umbral);

        if(is_string($pred)){
            $pred = 'nada';
        }else{

            if($pred >= 5){
                $pred = 5;
                $query = "SELECT title FROM movies WHERE id = '$mov'";
                $res = $con->query($query);
                while($row = $res->fetch_assoc()){
                    $title = $row['title'];
                }

                array_push($predictions, array($mov,$title,$pred));

                $counter += 1;
            }
        }
    }
    $order = array_column($predictions, 0);
    array_multisort($order, SORT_DESC, $predictions);
    array_splice($predictions, $limit);
    return $predictions;

}
```

cuestión. Es por esto que realizamos en item-similitude la "query" que nos permite optimizar un poco la tarea, pero aun así para la realización de pruebas ha sido necesaria la implementación de límites en las "queries" ya que no es factible la realización de este cálculo con la totalidad de la lista de películas sin, al menos, almacenar la lista de similitudes en una base de datos o similar.

Gestión de un nuevo usuario

Comparado con los previos apartados, la gestión de un nuevo usuario fue simple al poder reutilizar las funciones de las recomendaciones User-User para el ranking y para la predicción, necesitando tan solo incorporar pequeñas variaciones para que funcionasen con el usuario 0. Tanto “gestion.php” como “gestion.js” son muy similares a “useruser.php” y “useruser.js”, con la única variación real siendo el script que usamos para votar una película, “rate.php”.

```
movie = $_REQUEST['movieid'];
rating = $_REQUEST['rating'];

sql_query = "SELECT DISTINCT movieid from ratings WHERE movieid='$movie' and userid='0'";
result = $con->query($sql_query);
f (mysqli_num_rows($result) != 0){
    echo "El usuario ha votado esta película en el pasado";
else{
    $sql_query = "INSERT INTO ratings(userid, movieid, rating) VALUES ('0', $movie, $rating)";
    if($con->query($sql_query) === TRUE){
        echo "El usuario ha votado la película ",$movie," con un ",$rating;
    }else{
        echo "Error inseperado";
    }
```

Como la tabla ratings no admite que un mismo usuario haya votado varias veces a la misma película, lo primero que hacemos es una comprobación de que el usuario no haya votado ya a la película que está intentando valorar. Si lo ha hecho, el programa simplemente le da un mensaje de error.

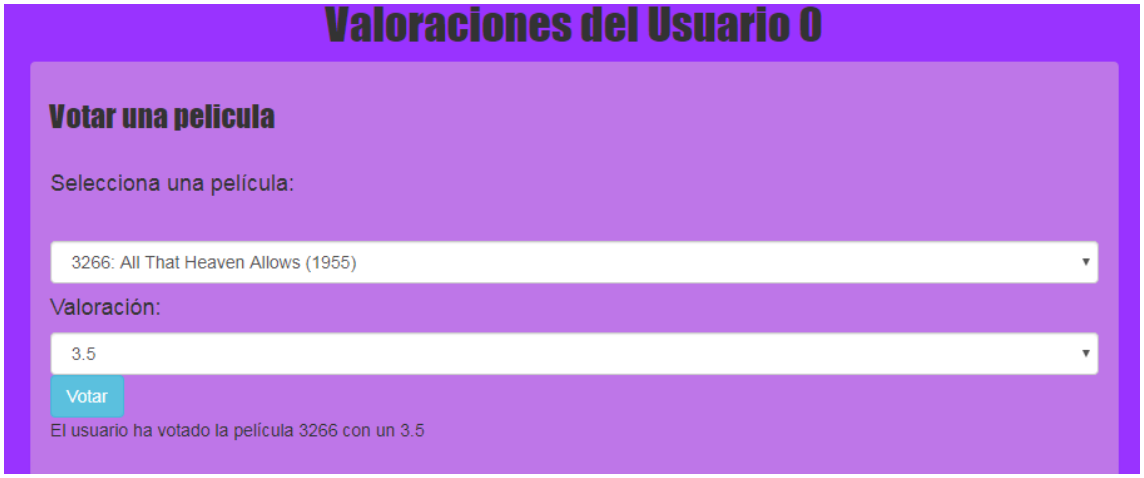
Si no lo ha hecho, entonces se efectúa la votación correctamente. Cabe decir que para que puedan realizarse tanto el ranking como la predicción, el usuario 0 debe votar a múltiples películas, o de lo contrario ningún usuario tendrá una similitud lo suficientemente elevada.

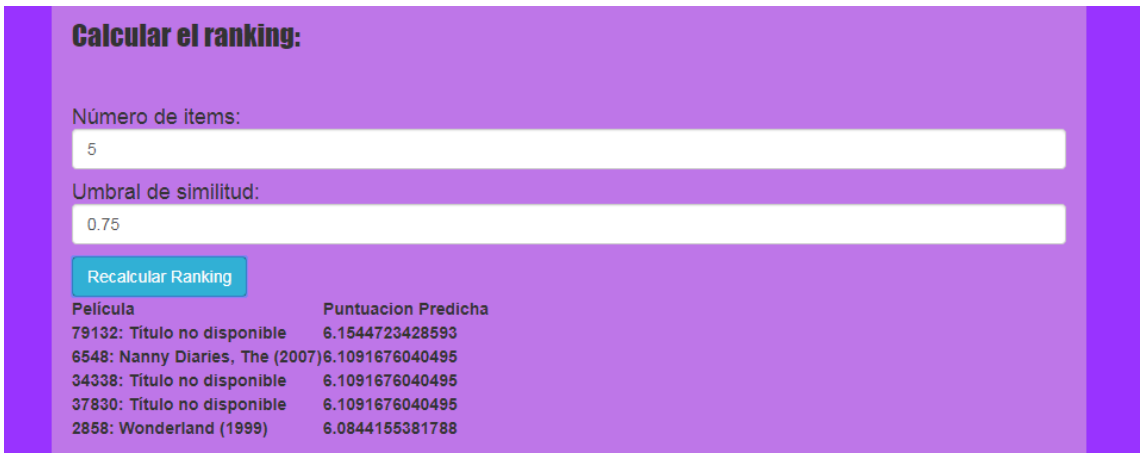
Casos de Uso

Cuando votamos una película, tenemos que seleccionarla y la puntuación que le vamos a dar. Nos mostrará un mensaje confirmando que nuestra puntuación se ha registrado correctamente.

Como MovieLens no permite múltiples valoraciones, si se vota a la misma película (da igual la puntuación), la página nos dará un mensaje de error.

Cuando mostramos el ranking del usuario cero (siendo importante haber votado suficientes películas como para poder tener valores de similitud suficientemente altos), podemos ver que al votar una película de él, nuestro ranking se actualizará y ya no lo incluye.





Descripción del portal web

Nuestra página no está subida a remoto por lo que no tiene URL de acceso, y sólo funciona en local por lo tanto para acceder a ésta una vez se haya colocado la carpeta del proyecto en la carpeta de nuestro servidor apache debemos usar la dirección localhost/RecommenderSystem/Código o el nombre de la carpeta en cuestión. (en el siguiente apartado hablaremos del manual de instalación necesario para replicar nuestro entorno y la página).

Nuestro index es muy simple, pues tan solo nos permite acceder a los tres apartados principales del proyecto, es decir, las recomendaciones User-User, las recomendaciones Item-Item, y la gestión del nuevo usuario.

Para entender mejor los detalles de estas partes, puede leer los apartados de Casos de Uso.

Como lo único que cambia entre User-User e Item-Item es el back-end (es decir, las funciones para calcular las predicciones), son idénticas desde el punto de vista del usuario. En ambos casos, tenemos la posibilidad de calcular el ranking de películas seleccionando al usuario que queramos, y editar los valores de umbral de similitud (es decir, a partir de qué coeficiente de correlación consideramos que un usuario o ítem debe pertenecer al vecindario) y el número de películas que saldrán en el ranking. Mientras se calcula, nos mostrará un mensaje para saber que lo está haciendo (porque, como se ha dicho, las operaciones pueden ser muy largas).

El otro apartado que incluyen es el de predicción, en el que simplemente escogemos un usuario, una película, y también el umbral de similitud. El umbral de similitud es editable (aunque no fuera un requisito) para permitir que aquellos con ordenadores o servidores más lentos puedan escoger un umbral más alto y reducir el tiempo de ejecución.

Por último, la gestión de usuario funciona exactamente igual que User-User, solo que no se selecciona nunca el usuario porque siempre se considera que se ha seleccionado al usuario 0. Lo único nuevo es el apartado de votar una película, donde el usuario debe seleccionar cualquiera de los valores aceptados para MovieLens (0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 y 5).

Manual de instalación

Para comenzar con la instalación, deberemos de insertar nuestra carpeta del proyecto en la carpeta de nuestro hosting local. Una vez tengamos localizada la carpeta de nuestro proyecto,

dentro de esta habrá un subdirectorio de nombre “DB”. En este se encontrarán los dos scripts necesarios para configurar la base de datos, con nombre DBScript y CSVScript.

Para poder utilizar estos scripts será necesario instalar en su interpretador de Python las librerías “mysql.connector” y “pandas”.

Deberemos lanzar primero DBScript para crear la base de datos ya que si lanzamos primero CSVScript no funcionará sin haber lanzado el otro primero. Una vez creada entonces lanzaremos el CSVScript para rellenar la base de datos con la información de los ficheros con extensión .csv .

Una vez generada la base de datos con la información correspondiente, abriremos un explorador e introduciremos la dirección como se comentó en el apartado anterior.

Una vez abierto ya podrá disfrutar de los servicios de RecommenderSystem.

Bibliografía

https://www.php.net/manual/es/function.array-multisort.php. (s.f.).

https://www.socscistatistics.com/tests/pearson/. (s.f.).

https://www.w3schools.com/sql/default.asp

https://pandas.pydata.org/