



UNIVERSIDAD DE GRANADA

PRÁCTICA 2: DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN BASADO EN FILTRADO COLABORATIVO

Autor

Juan Manuel Castillo Nievas



MÁSTER PROFESIONAL EN INGENIERÍA INFORMÁTICA 2020/2021

Granada, 10 de abril de 2021

Índice

1. Descripción inicial	2
2. Logística	2
3. Recomendaciones basadas en filtrado colaborativo	3
3.1. Cálculo de la similitud con Correlación de Pearson	5
3.2. Cálculo de la similitud con Coseno	5
4. Implementación	6
4.1. Clase Users	6
4.2. Clase Movies	6
4.3. Clase RecommendationSystem	7
4.3.1. Método createRatingMatrix	7
4.3.2. Método calculateSimilarity	8
4.3.3. Método calculateNeighborhood	9
4.3.4. Método getRecommendations	9
4.3.5. Método pearsonCorrelation	10
4.3.6. Método cosine	11
5. Manual de usuario	12
6. Referencias	16

1. Descripción inicial

Durante esta práctica se ha desarrollado un **Sistema de Recomendación basado en Filtrado Colaborativo**. Este sistema muestra inicialmente al usuario 20 películas aleatorias para que este las evalúe, asignando 1 estrella cuando no le gusta nada, hasta 5 estrellas indicando que es una de sus películas favoritas.

Una vez se obtienen las valoraciones del usuario, el sistema calculará la **similitud entre el usuario y el resto de usuarios**, obteniendo el conjunto de vecinos (usuarios) que más se parecen a él en cuanto a películas vistas y valoraciones.

El objetivo de esta aplicación es **predecir** la valoración de todas las películas que el usuario no ha visto pero que sí han visto sus vecinos más cercanos, mostrando aquellas películas que se predicen con una **valoración mayor o igual a 4 estrellas**.

El lenguaje elegido ha sido **Python** y como **editor de texto** para su desarrollo se ha elegido **Sublime Text 3** [1].

2. Logística

Para el desarrollo de este sistema se ha trabajado con la colección **MovieLens** [2], compuesta por **100.000** valoraciones de **943 usuarios** sobre **1682 películas**.

El fichero **u.data** contiene todas las valoraciones con el siguiente formato:

```
idUser idMovie Valoración Timestamp
```

El fichero **u.item** contiene todas las películas con el siguiente formato:

```
idMovie|titulo|...
```

Para el desarrollo de este sistema sólo hace falta tener en cuenta **idUser**, **idMovie**, **Valoración** y **titulo**.

3. Recomendaciones basadas en filtrado colaborativo

Antes de entrar en la implementación del sistema, se va a explicar en qué consiste el filtrado colaborativo y las métricas de similitud usadas para entender qué es lo que se ha implementado.

El filtrado colaborativo tiene dos enfoques: basado en el usuario y basado en artículos. En este caso, se ha usado el basado en usuario que se basa en la **similitud o vecindario del usuario**.

Primero, se buscan a los usuarios que sean similares, es decir, aquellos usuarios que compartan valoraciones en las mismas películas que el usuario objetivo y que además coincidan en la valoración. Si dos usuarios son similares, se pueden recomendar películas que uno de ellos haya visto y que el otro no.

Para obtener los usuarios similares se utiliza lo que se conoce como **matriz de votos**. En las filas se representan a los usuarios, en las columnas las películas, y en la intersección se pone la valoración que ha dado el usuario para la película. Un ejemplo de **matriz de votos** se propone en el ejemplo de la Figura 1 [3]. Se tienen las calificaciones de 4 usuarios para 5 películas. El usuario objetivo ha valorado 3 de ellas.

Cuando se tiene la matriz de votos, el siguiente paso es **calcular la similitud con los demás usuarios**. Este cálculo se ha hecho utilizando la **Correlación de Pearson** y la **similitud del Coseno**, que se explicarán más adelante. En el ejemplo propuesto, el usuario objetivo presenta una similitud muy alta (del 0.9) con el usuario de la fila 2.

Una vez se tiene calculada la similitud entre los usuarios, se procede a calcular una **posible valoración del usuario** acerca de las demás películas sólo teniendo en cuenta las valoraciones de los k vecinos más similares. Esto se calcula con la siguiente fórmula:

$$\bar{r}(u, i) = \frac{\sum sim(u, v) * r(v, i)}{\sum |sim(u, v)|}$$

En el numerador se hace la sumatoria de la similitud del usuario objetivo con sus vecinos multiplicado por la valoración del usuario, y en el denominador se hace la sumatoria de la similitud del usuario objetivo con sus vecinos.

Por ejemplo, en nuestro ejemplo actual (Figura 1), si tenemos en cuenta $k = 2$ (los dos vecinos más similares) y queremos hacer una predicción de la quinta película, la predicción sería la siguiente (en este ejemplo las valoraciones van desde 1 estrella hasta 10 estrellas):

$$\bar{r}(u, i) = \frac{(8 * 0,9) + (0,7 * 7)}{0,9 + 0,7} = 7,56$$

Por último, el final sería **recomendar aquellas películas cuya valoración calculada haya sido superior a 4 estrellas**, indicando que tiene alta probabilidad de que le guste al usuario objetivo.



Figura 1: Matriz de votos [3]

3.1. Cálculo de la similitud con Correlación de Pearson

Para el cálculo de la similitud se ha usado la **Correlación de Pearson**, cuya fórmula es la siguiente:

$$sim(u, v) = \frac{\sum(r(u, i) - \bar{r}(u))(r(v, i) - \bar{r}(v))}{\sqrt{\sum(r(u, i) - \bar{r}(u))^2} \sqrt{\sum(r(v, i) - \bar{r}(v))^2}}$$

El problema de esta fórmula es que puede darse el caso de que el denominador sea 0. Por ejemplo, la media de valoraciones del usuario objetivo, $\bar{r}(u)$, es 4, y ha dado una valoración $r(u, i)$ de 4, entonces el denominador se haría 0.

La **solución** a esto ha sido calcular la **similitud del Coseno** para estos casos especiales, porque dicha similitud no resta de la media y por lo tanto nunca se hará 0 el denominador. El problema es que se obtienen unas similitudes con **Pearson**, cuyo rango va desde **-1 hasta 1** y otras con el **Coseno**, cuyo rango va desde **0 hasta 1**. Lo que se ha hecho ha sido **normalizar todas las similitudes obtenidas de Pearson**.

3.2. Cálculo de la similitud con Coseno

A pesar de ser menos robusta, esta similitud se ha utilizado cuando el de **Pearson** ha obtenido un denominador de 0, tal y como se ha comentado anteriormente. Su fórmula es la siguiente:

$$sim(u, v) = \frac{\sum(r(u, i))(r(v, i))}{\sqrt{\sum(r(u, i))^2} \sqrt{\sum(r(v, i))^2}}$$

La fórmula es muy similar a la de Pearson, lo único que cambia es que no se tiene en cuenta la media de los usuarios.

4. Implementación

4.1. Clase Users

Esta clase se encuentra en el archivo **users.py** y representa a los usuarios. Está formado por un diccionario en el que **la clave es idUser**, y el valor es un vector en el que cada elemento es un diccionario en el que **la clave es idMovie** y el valor es la valoración del usuario para esa película. De forma visual, la estructura sería la siguiente:

```
{ idUser: { [{idMovie: rating}, {idMovie2: rating}] },  
  idUser2: { [{idMovie2: rating}, {idMovie5: rating}, {idMovie6: rating}] } }
```

La clase está compuesta por tres métodos:

- **__init__**: es el constructor de la clase. Crea un diccionario de usuarios vacío.
- **addUser**: añade un usuario. Si el usuario no existe, lo crea. Si el usuario existe, añade al usuario la valoración de la película.
- **getUsers**: devuelve el diccionario de usuarios. Aunque no sea necesario porque los atributos son públicos, se ha hecho para una mayor limpieza visual de código cuando se utiliza.

```
1 class Users:  
2     def __init__(self):  
3         ...  
4  
5     def addUser(self, idUser, idMovie, rating):  
6         ...  
7  
8     def getUsers(self):  
9         ...
```

4.2. Clase Movies

Esta clase se encuentra en el archivo **movies.py** y representa a las películas. Está formado por un diccionario en el que **la clave es idMovie** y **el valor es title**. De forma visual, la estructura sería la siguiente:

```
{ idMovie: title,  
  idMovie2: title2 }
```

De forma análoga a la clase **Users**, la clase está compuesta por tres métodos:

- `__init__`: es el constructor de la clase. Crea un diccionario de películas vacío.
- `addMovie`: añade una película con el id y el título al diccionario.
- `getMovies`: devuelve el diccionario de películas.

```
1 class Movies:
2     def __init__(self):
3         ...
4
5     def addMovie(self, idMovie, title):
6         ...
7
8     def getMovies(self):
9         ...
```

4.3. Clase RecommendationSystem

Esta clase es la encargada de recibir el conjunto de datos de usuarios y películas y hacer todo el proceso de recomendación basado en filtrado colaborativo. El constructor de la clase es el siguiente:

```
1 class RecommendationSystem:
2     def __init__(self, users, movies):
3
4         self.users = users
5         self.movies = movies
```

El constructor recibe los usuarios y películas anteriormente creadas a partir del conjunto de datos de **MovieLens**.

A continuación se va a proceder a explicar los métodos de la clase usados en el proceso de recomendación.

4.3.1. Método createRatingMatrix

Este método crea una **matriz de votos**. En las filas se representan los usuarios y en las columnas las películas, y en la intersección se indica la valoración que ha hecho un usuario para una película. Si el usuario no ha tenido valoración para una película, se indica con **NaN**.

```
1 def createRatingMatrix(self):
2     self.matrix = np.empty((len(self.users.getUsers()), len(self.movies.getMovies())))
3     self.matrix[:] = np.nan
4
5     for user in self.users.getUsers():
6         for i in range(len(self.users.getUsers()[user])):
```



```

7         idUser = user
8         idMovie = list(self.users.getUsers()[user][i].keys())[0]
9         rating = list(self.users.getUsers()[user][i].values())[0]
10
11         self.matrix[idUser][idMovie-1] = rating

```

4.3.2. Método calculateSimilarity

Este método calcula la similaridad del usuario objetivo con el resto de usuarios. Este método obtiene como salida una matriz formada por dos columnas: la primera representa el id del usuario y la segunda la similaridad obtenida con dicho usuario.

El algoritmo seguido es el siguiente:

1. Se crea la matriz de votos
2. Se recorren todos los usuarios
 - a) Se recorren todas las películas. Para cada película:
 - 1) Si el usuario objetivo la ha valorado y el usuario actual también, se añade al vector de películas en común
 - b) Si el usuario objetivo no tiene ninguna película en común, se pasa al siguiente usuario. Si hay películas en común, se procede a calcular la similitud:
 - 1) Se calcula la similitud con la Correlación de Pearson.
 - 2) Si el denominador de Pearson es 0, se calcula la similitud con el coseno.
3. Se ordenan los usuarios poniendo en primera posición aquellos que tengan una mayor similitud

```

1  def calculateSimilarity(self):
2      self.createRatingMatrix()
3
4      n_users = self.matrix.shape[0]
5      my_user_ratings = self.matrix[0]
6      my_avg = np.nanmean(self.matrix[0], axis=0)
7      similarity_matrix = []
8
9      for user in range(1, n_users):
10         user_ratings = self.matrix[user]
11         user_avg = np.nanmean(user_ratings)
12
13         common_movies = []
14
15         for movie, my_rating in enumerate(my_user_ratings):
16             if not np.isnan(my_rating):
17                 user_rating = self.matrix[user][movie]
18                 if not np.isnan(user_rating):
19                     common_movies.append(movie)

```

```

20
21     if len(common_movies) == 0:
22         continue
23
24     sim = self.pearsonCorrelation(common_movies, my_user_ratings, user_ratings, my_avg,
25                                   user_avg)
26
27     if sim is None:
28         sim = self.cosine(common_movies, my_user_ratings, user_ratings)
29
30     if sim > 1:
31         sim = 1.0
32
33     similarity_matrix.append([user,sim])
34
35 self.similarity_matrix = sorted(similarity_matrix,key=lambda x: x[1], reverse=True)

```

4.3.3. Método calculateNeighborhood

Este método es muy sencillo y simplemente se encarga de elegir los k vecinos más cercanos, es decir, los usuarios más similares al usuario objetivo.

```

1 def calculateNeighborhood(self, k):
2
3     self.neighborhood = []
4     for i in range(k):
5         self.neighborhood.append([self.similarity_matrix[i][0],self.similarity_matrix[i][1]])

```

4.3.4. Método getRecommendations

Este método calcula las recomendaciones del usuario. El algoritmo es el siguiente:

1. Se recorren todas las películas:
 - a) Si el usuario no ha valorado una película:
 - 1) Si los vecinos han valorado dicha película, se calcula la predicción
 - 2) Se guarda la predicción en las recomendaciones si el valor predicho es mayor o igual a 4
2. Se ordenan las recomendaciones de mayor predicción a menor

```

1 def getRecommendations(self):
2     my_user_ratings = self.matrix[0]
3     recommendations = []

```

```

4
5     for movie, my_rating in enumerate(my_user_ratings):
6         numerator = 0
7         denominator = 0
8         if np.isnan(my_rating):
9             for user in self.neighborhood:
10                 idUser = user[0]
11                 similarity = user[1]
12                 user_rating = self.matrix[idUser][movie]
13
14                 if not np.isnan(user_rating):
15                     numerator += similarity*user_rating
16                     denominator += similarity
17
18         if not denominator == 0:
19             rating = numerator/denominator
20
21         if rating >= 4:
22             recommendations.append([movie+1, rating])
23
24     recommendations = sorted(recommendations, key=lambda x: x[1], reverse=True)
25
26     return recommendations

```

4.3.5. Método pearsonCorrelation

Este método calcula la similitud entre dos usuarios usando la **Correlación de Pearson**. Muy importante normalizar el resultado, ya que en caso de que el denominador sea 0, se va a usar la similitud del Coseno.

```

1 def pearsonCorrelation(self, common_movies, my_user_ratings, user_ratings, my_avg,
2     user_avg):
3     numerator = 0
4     denominator1 = 0
5     denominator2 = 0
6
7     for movie in common_movies:
8         numerator += (my_user_ratings[movie]-my_avg)*(user_ratings[movie]-user_avg)
9         denominator1 += pow((my_user_ratings[movie]-my_avg),2)
10        denominator2 += pow((user_ratings[movie]-user_avg),2)
11
12    denominator = sqrt(denominator1)*sqrt(denominator2)
13
14    if denominator == 0:
15        return None
16
17    else:
18        sim = numerator/denominator

```

```
18     sim_normalized = (sim - (-1)) / 2
19
20     return sim_normalized
```

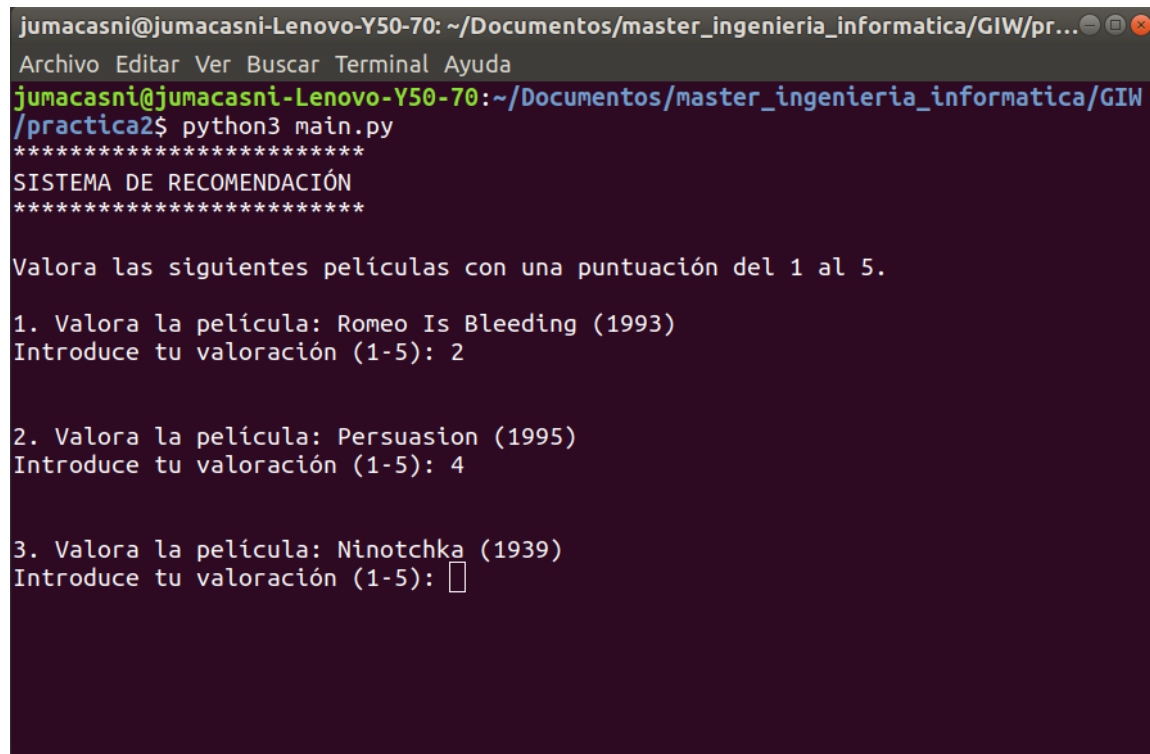
4.3.6. Método cosine

Este método calcula la similitud entre dos usuarios usando la **similitud del Coseno**.

```
1 def cosine(self, common_movies, my_user_ratings, user_ratings):
2     numerator = 0
3     denominator1 = 0
4     denominator2 = 0
5
6     for movie in common_movies:
7         numerator += my_user_ratings[movie]*user_ratings[movie]
8         denominator1 += pow(my_user_ratings[movie],2)
9         denominator2 += pow(user_ratings[movie],2)
10
11     denominator = sqrt(denominator1)*sqrt(denominator2)
12
13     return numerator/denominator
```

5. Manual de usuario

El archivo **main.py** contiene el programa principal. Se debe ejecutar usando `python3 main.py` y los archivos del conjunto de datos (`u.data` y `u.item`) deben estar en el mismo directorio. Cabe destacar que en esta sistema, el número de vecinos que forman el vecindario es de 10. Al ejecutarlo, se mostrarán inicialmente 20 películas al azar que se deben valorar (ver Figura 2).



```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW/practica2$ python3 main.py
*****
SISTEMA DE RECOMENDACIÓN
*****

Valora las siguientes películas con una puntuación del 1 al 5.

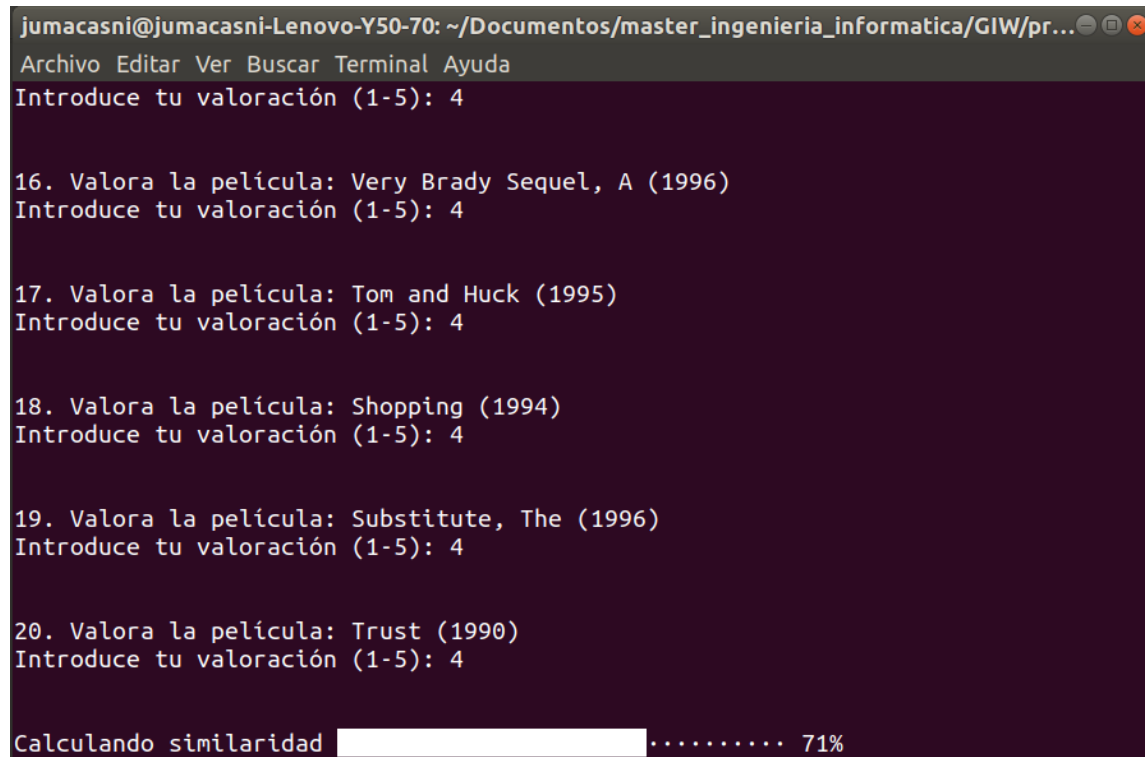
1. Valora la película: Romeo Is Bleeding (1993)
Introduce tu valoración (1-5): 2

2. Valora la película: Persuasion (1995)
Introduce tu valoración (1-5): 4

3. Valora la película: Ninotchka (1939)
Introduce tu valoración (1-5): 
```

Figura 2: El usuario debe introducir la valoración de 20 películas inicialmente

Una vez se han indicado las 20 valoraciones, se va a calcular la similitud con el resto de usuarios. Como este proceso puede tardar un poco, se muestra un *feedback* al usuario, tal y como se muestra en la Figura 3.



```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
Introduce tu valoración (1-5): 4

16. Valora la película: Very Brady Sequel, A (1996)
Introduce tu valoración (1-5): 4

17. Valora la película: Tom and Huck (1995)
Introduce tu valoración (1-5): 4

18. Valora la película: Shopping (1994)
Introduce tu valoración (1-5): 4

19. Valora la película: Substitute, The (1996)
Introduce tu valoración (1-5): 4

20. Valora la película: Trust (1990)
Introduce tu valoración (1-5): 4

Calculando similitud [Progress Bar] ..... 71%
```

Figura 3: Cálculo de similitudes

Cuando el programa ha terminado de calcular las similitudes, se pasa al método que obtiene las recomendaciones. Se le muestra al usuario el número de recomendaciones encontradas y si desea verlas (ver Figura 4).

```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
17. Valora la película: Tom and Huck (1995)
Introduce tu valoración (1-5): 4

18. Valora la película: Shopping (1994)
Introduce tu valoración (1-5): 4

19. Valora la película: Substitute, The (1996)
Introduce tu valoración (1-5): 4

20. Valora la película: Trust (1990)
Introduce tu valoración (1-5): 4

Calculando similitud  100%

Obteniendo recomendaciones  100%

Hay 411 películas recomendadas para ti, ¿quieres verlas? (s/n):

```

Figura 4: Número de recomendaciones

La salida final del programa es una lista ordenada de mayor valoración precedida a menor de las películas recomendadas al usuario (ver Figura 5).

A screenshot of a terminal window with a dark background and light-colored text. The window title bar shows the user 'jumacasni' on a 'Lenovo-Y50-70' machine, with the path '~/Documentos/master_ingenieria_informatica/GIW/pr...'. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal content shows a progress bar at 100% for 'obteniendo recomendaciones'. It then asks 'Hay 411 películas recomendadas para ti, ¿quieres verlas? (s/n):' and receives the input 's'. Below this, it says 'Recomendaciones:' followed by a list of movies and their ratings, all of which are 5.0. The list includes 'Richard III (1995)', 'Usual Suspects, The (1995)', 'Taxi Driver (1976)', 'Belle de jour (1967)', 'Net, The (1995)', 'Madness of King George, The (1994)', 'Priest (1994)', 'Three Colors: White (1994)', and 'Carlito's Way (1993)'.

```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
obteniendo recomendaciones 100%

Hay 411 películas recomendadas para ti, ¿quieres verlas? (s/n):
s

Recomendaciones:

0. Richard III (1995) --> Valoración: 5.0
1. Usual Suspects, The (1995) --> Valoración: 5.0
2. Taxi Driver (1976) --> Valoración: 5.0
3. Belle de jour (1967) --> Valoración: 5.0
4. Net, The (1995) --> Valoración: 5.0
5. Madness of King George, The (1994) --> Valoración: 5.0
6. Priest (1994) --> Valoración: 5.0
7. Three Colors: White (1994) --> Valoración: 5.0
8. Carlito's Way (1993) --> Valoración: 5.0
```

Figura 5: Películas recomendadas

6. Referencias

- [1] Sublime text 3. <https://www.sublimetext.com/>.
- [2] Movielens. <https://grouplens.org/datasets/movielens/>.
- [3] Stat Dev. Recomendaciones basado en filtrado colaborativo. <https://www.statdeveloper.com/recomendaciones-basado-en-filtrado-colaborativo/>.