Máster Universitario en Ingeniería Informática

Inteligencia Computacional

GESTIÓN DE INFORMACIÓN EN LA WEB

Práctica 2:

Desarrollo de un Sistema de Recomendación basado en Filtrado Colaborativo

UNIVERSIDAD DE GRANADA

Carlos Morales Aguilera
75925767-F
carlos7ma@correo.ugr.es

Curso Académico 2020-2021

Índice

1. Descripción de la práctica **2**

2. Estructura del proyecto **2**

3. Sistema de recomendación basado en filtrado colaborativo **2**
3.1. Constructor (___init___) 3
3.2. set_initial_ratings 5
3.3. calculate_neighborhood 6
3.4. get_neighborhood 7
3.5. calculate_recommendations 7
3.6. get_recommendations 9
3.7. print_recommendations 9
3.8. export_recommendations 9
3.9. see_recommendations 10

4. Programa principal **11**

5. Innovaciones **12**

6. Manual de uso **13**

7. Referencias Bibliográficas **16**

1. Descripción de la práctica

La práctica a resolver consiste en la elaboración y desarrollo de un Sistema de Recomendación basado en Filtrado Colaborativo, utilizando para ello la colección de películas **MovieLens**, la cual contiene 100.000 valoraciones de 943 usuarios sobre 1682 películas.

El lenguaje de programación escogido es el de Python, ya que es un lenguaje sencillo y con un gran soporte de cara a las diferentes operaciones que se requieren hacer, y posee un gran soporte. Para el procesamiento de ficheros se utilizará la biblioteca **Pandas** y para operaciones sobre arrays y matrices se utilizará la biblioteca **Numpy**.

La idea de la práctica consistirá en realizar una aplicación que obtenga las valoraciones de 20 películas por parte del usuario, a continuación obtenga el vecindario (de 10 vecinos) mediante una matriz de valoraciones y la correlación entre los usuarios con el usuario principal por sus valoraciones. Por último a partir de esta correlación y las valoraciones de los vecinos obtener las películas con una valoración mayor de 4 estrellas.

Toda la lógica de la aplicación se contendrá para la modularización en una clase principal que controle las diferentes funcionalidades y que permita también exportar los resultados obtenidos del usuario.

2. Estructura del proyecto

El proyecto se compone de 2 ficheros principales:

- **recommendation_system.py**: Se implementa la clase **RecommendationSystem**, la cual contiene toda la lógica de la aplicación del sistema de recomendación mediante filtrado colaborativo de forma encapsulada para únicamente hacer uso de esta.
- **main.py**: Programa principal donde se realiza el procedimiento de las diferentes partes de la práctica llamando a la clase previamente descrita.

3. Sistema de recomendación basado en filtrado colaborativo

La clase **RecommendationSystem** es la clase principal donde se contiene toda la lógica de la aplicación. Para ello contiene diferentes métodos y atributos que permiten englobar toda esta funcionalidad, de forma que se pueda crear un sistema robusto y sencillo, y sobre todo fácil de usar por cualquier usuario. Para ello a continuación se explican los siguientes métodos:

3.1. Constructor (___init___)

El constructor principal de la clase contiene todos los elementos necesarios para el sistema de recomendación, siendo estos:

- **self.___ratings**: Las valoraciones de todos los usuarios que se leen del fichero que se le indique al sistema. Para ello hace uso de la biblioteca **Pandas** y lee el fichero CSV asociado.
- **self.___movies**: Las películas existentes que se leen del fichero que se le indique al sistema. Se leen de la misma forma que las valoraciones.
- **self.___num_to_rate**: Número de películas que deben ser valoradas por el usuario de forma aleatoria.
- **self.___num_neighborhood**: Número de vecinos que se deben calcular en el vecindario.
- **self.___neighborhood**: Vecindario del usuario actual, inicialmente vacío.
- **self.___user_ratings**: Valoraciones del usuario, inicialmente vacías.
- **self.___recommendations**: Recomendaciones que se realizan al usuario, inicialmente vacías.
- **self.___export**: Fichero donde realizar la exportación en caso de que el usuario lo desee.

```
def __init__(self, ratings_file: str, movies_file: str, num_to_rate: int,
    neighborhood: int, export: str):
  # Read ratings from ratings_file path
  ratings = ["idUser","idMovie","rating","timestamp"]
  self.__ratings = read_csv(ratings_file,
     engine='python',
     sep='\t',
     header=None,
     names=ratings)
  # Read movies from movies_file path
  movies = ["idMovie","title","date"]
  self.__movies = read_csv(movies_file,
     engine='python',
     sep='|',
     usecols=range(0,3),
     header=None,
     names=movies)
  # Number of movies to be rated by the user
  self.__num_to_rate = num_to_rate
  # Number of the neighborhood to be generated
  self.__num_neighborhood = neighborhood
  # User neighborhood
  self.__neighborhood = {}
  # User ratings
  self.__user_ratings = {}
  # User recommendations
  self.__recommendations = {}
  # File to export
  self.__export = export
```

3.2. set_initial_ratings

El método sirve para recoger las valoraciones iniciales del usuario, para ello escoge de forma aleatoria 20 películas aleatorias del conjunto de películas y por consola recibe las valoraciones del usuario, comprobando y forzando a que sea de 1 a 5 estrellas:

```
def set_initial_ratings(self):
   # Get num_to_rate random movies to rate by user
   rated_movies = self.__movies.sample(self.__num_to_rate)

   print(f"\n\nUSER RATINGS: Please rate each movie from 1 to 5 stars.\n\n")

   # Define a count
   count = 1
   # Let user rate each movie
   for movie in rated_movies["idMovie"]:
      # Get movie title
      movie_title =
          rated_movies[rated_movies["idMovie"]==movie]["title"].values[0]

      while True:
         print(f"{count}. {movie_title}")
         # Get user input as rating
         rating = int(input())
         # Check rating is valid
         if rating not in range(1,6):
            print("Input rating is not valid, ratings must be between 1 and 5
                stars.")
         else:
            break

      # Save rating associated to movie
      self.__user_ratings[movie] = rating
      count += 1
```

3.3. calculate_neighborhood

El método sirve para calcular el vecindario, para ello se crea una matriz donde se recogen las valoraciones de cada usuario a cada una de las películas, de esta forma en un una matriz se recogerían todas las valoraciones y se marcaría como 0 si un usuario no ha marcado una determinada pregunta.

A continuación se calculan los coeficientes de correlación y se crea una matriz de correlación para los diferentes usuarios del sistema, con eso se obtiene la correlación entre dos usuarios, para determinar su similitud.

Por último se obtiene solo la primera columna que es la correspondiente al usuario que ejecuta el programa y se obtiene un vector de correlación entre el usuario con cada uno de los usuarios del sistema, y se ordena para obtener aquellos usuarios con mayor correlación para crear el vecindario.

```
def calculate_neighborhood(self):
   # Group users by ID into a set
   users = set(self.__ratings["idUser"])
   # Add rating user as 0
   users.add(0)

   # Create a matrix that groups the users and movies
   ratings_matrix = DataFrame(index=users, columns=self.__movies["idMovie"])
   # Fill all the matrix with 0
   ratings_matrix = ratings_matrix.fillna(0)
   # Add user ratings to matrix
   for movie in self.__user_ratings.keys():
      ratings_matrix.iloc[0, movie-1] = self.__user_ratings[movie]
   # Add rest of users ratings to matrix
   for user in users:
      # Get user ratings
      user_ratings = self.__ratings[self.__ratings["idUser"] == user]
      # Add user ratings to matrix
      for movie in user_ratings["idMovie"]:
         ratings_matrix.iloc[user, movie-1] =
             user_ratings[user_ratings["idMovie"]==movie]["rating"].values[0]

   # Calculate correlation coefficients matrix from ratings matrix
   corr_matrix = corrcoef(ratings_matrix)
   # The first column contains the comparison between actual user and rest of
       users
   corr = corr_matrix[0]
   # Sort indexes (users id) in descending order (from best to worst)
   users_order = argsort(corr)[::-1]

   # Create the neighborhood by selecting the n-best users compared to actual
       user
   for i in range(self.__num_neighborhood):
      self.__neighborhood[users_order[i+1]]=corr[users_order[i+1]]
```

3.4. get_neighborhood

Este método simplemente devuelve el vecindario generado, por si lo quiere consultar el usuario.

```
def get_neighborhood(self):
   return self.__neighborhood
```

3.5. calculate_recommendations

Este método es el encargado de realizar las recomendaciones al usuario. Para ello obtiene las películas valoradas por el usuario, y los usuarios del vecindario. Calcula la media de las valoraciones de cada uno de estos usuarios.

Una vez se obtiene la información y se calcula el grado de recomendación o valoración posible del usuario aplicándo la fórmula:

$$\hat{r}(u,i) = \overline{r}(u) + C\sum sim(u,v)(r(v,i) - \overline{r}(v))$$

donde

$$C = \frac{1}{\sum |sim(u,v)|}$$

Para ello se utiliza por cada película y cada vecino, si ha valorado la película, el grado de correlación con el usuario, junto a la valoración y la media de valoración del usuario.

Por último, solo se consideran valoraciones que consideren 4 estrellas o más y se almacenan dichas recomendaciones.

```
def calculate_recommendations(self):
   # Get user ratings
   rated_movies = self.__user_ratings.keys()
   # Calculate mean of user ratings
   user_mean = mean(list(self.__user_ratings.values()))
   # Calculate mean of user neighborhood ratings
   neighborhood_mean = {}
   for neighbour in self.__neighborhood.keys():
      neighbour_rating = self.__ratings[self.__ratings["idUser"] == neighbour]
      neighbour_mean = neighbour_rating["rating"].mean()
      neighborhood_mean[neighbour] = neighbour_mean

   # Search every possible movie
   for movie in array(self.__movies["idMovie"]):
      # Search movies not rated by user
      if movie not in rated_movies:
         sim, corrC = 0, 0
         # Get neighborhood rating
         for neighbour in self.__neighborhood.keys():
            # Get neighbour correlation
            corr = self.__neighborhood[neighbour]
            # Get movie rating by neighbour
            movie_rating = self.__ratings[(self.__ratings["idMovie"]==movie) &
                (self.__ratings["idUser"]==neighbour)]
            # Check that neighbour has rated the movie
            if not movie_rating.empty:
               # Get finally the movie rating by user
               movie_rating = movie_rating["rating"].values[0]
               # Sum into sim and corrC
               sim += corr*(movie_rating - neighborhood_mean[neighbour])
               corrC += abs(corr)

         # Calculate interest for user
         interest = 0
         if corrC > 0:
            # Calculate the interest value
            interest = user_mean + sim/corrC

            # Force higher interest to be 5 stars
            if interest > 5:
               interest = 5
         # Check if interest is higher than 4 starts (then it is a recommendation)
         if interest >= 4:
            self.__recommendations[movie]=interest

   # Sort recommendations in descending order from best to worst
   self.__recommendations = dict(sorted(self.__recommendations.items(),
       key=lambda item: item[1], reverse=True))
```

3.6. get_recommendations

Este método se encarga de devolver las recomendaciones con la estructura de un diccionario:

```
def get_recommendations(self):
   return self.__recommendations
```

3.7. print_recommendations

Este método se encarga de mostrar por consola la lista de recomendaciones de una forma natural para el usuario:

```
def print_recommendations(self):
   count = 1
   for movie in self.__recommendations.keys():
      title = self.__movies[self.__movies["idMovie"] == movie]["title"].values[0]
      print(f"{count}. {title}: {str(self.__recommendations[movie])}")
      count += 1
```

3.8. export_recommendations

Este método exporta las recomendaciones al fichero que indica el usuario al inicio del sistema de recomendaciones de forma natural para el usuario:

```
def export_recommendations(self):
   f = open(self.__export, "w")
   count = 1
   for movie in self.__recommendations.keys():
      title = self.__movies[self.__movies["idMovie"] == movie]["title"].values[0]
      f.write(f"{count}. {title}: {str(self.__recommendations[movie])}\n")
      count += 1
   f.close()
```

3.9. see_recommendations

Este método se encarga de preguntar al usuario si quiere ver y exportar las recomendaciones para que este decida indicándolo por consola y llamar a alguno de los métodos anteriores:

```
def see_recommendations(self):
   # See recommendations
   while True:
      # Ask user to print o export recommendations
      print(f"\nDo you want to see your recommendations? (y/n)")
      # Get user input
      user_input = input()
      if user_input == 'y':
         self.print_recommendations()
         break
      elif user_input == 'n':
         break

   # Export recommendations
   while True:
      # Ask user to print o export recommendations
      print(f"\nDo you want to export your recommendations? (y/n)")
      # Get user input
      user_input = input()
      if user_input == 'y':
         self.export_recommendations()
         print(f"\n\nRecommendations exported successfully on
             ..ml-data/u.recommendations")
         break
      elif user_input == 'n':
         break
```

4. Programa principal

El fichero **main.py** contiene la funcionalidad general del programa principal, el cual se encarga de simplemente definir los parámetros iniciales del sistema de recomendación anteriormente explicado y hacer llamadas a sus métodos para obtener la información asociada al vecindario y las recomendaciones que recibe el usuario.

```
from recommendation_system import RecommendationSystem

if(__name__=="__main__"):
   # Create a recommendation system with files paths
   # Rate 20 movies
   # Neighborhood of 10 neighbours
   system = RecommendationSystem(f"../ml-data/u.data", f"../ml-data/u.item", 20,
       10, "../ml-data/u.recommendations")

   # Rate 20 movies
   system.set_initial_ratings()

   # Calculate the neighborhood with initial ratings
   system.calculate_neighborhood()

   # Get the neighborhood
   neighborhood = system.get_neighborhood()

   # Print neighborhood
   print(f"The actual neighborhood for user is (user:similarity):\n
       {neighborhood}")

   # Calculate the recommendations by neighborhood
   system.calculate_recommendations()

   # Get the recommendations
   recommendations = system.get_recommendations()

   # Print the recommendations
   print(f"The recommendations for user is (idMovie:rating):\n
       {recommendations}\n")

   # Ask the user to print or save the recommendations
   system.see_recommendations()
```

5. Innovaciones

En cuanto a las innovaciones ofrecidas, se enumeran las siguientes:

- Se permite exportar las recomendaciones de forma personalizada en un fichero que indique el usuario mediante una ruta o *path*.
- Al utilizar las bibliotecas **Pandas** y **Numpy** la manipulación de CSVs y objetos de tipo array y matriz se vuelve extremadamente sencillo y eficiente, por lo que frente a otros posibles lenguajes de implementación, se facilita la manipulación y se realiza en un menor tiempo.
- Muchas de las implementaciones vistas durante la realización de la práctica utilizan modelos específicos, si bien no es una innovación, considero una mejora la abstracción de toda la funcionalidad en una clase reutilizable y fácilmente modificable para según que contexto.

6. Manual de uso

Al tratarse de un proyecto en *Python3*, es necesario tener definido el mismo, y con el fin de poder ejecutar el proyecto, es necesario instalar las bibliotecas de **Pandas** y **Numpy**, por lo que se podrían instalar sencillamente con el gestor de paquetes en Python **Pip**.

A continuación, es tan sencillo como ejecutar el programa principal de la manera:

```
python3 main.py
```

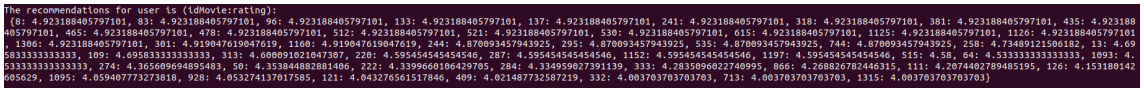
Una vez ejecutándose el programa mostrará una por una en forma de lista las películas para que cada una sea valorada individualmente por el usuario de la forma:

```
carlos@carlos-Zenbook:~/GIW/practica2_moralesaguilera_carlos/practica$ python3 main.py

USER RATINGS: Please rate each movie from 1 to 5 stars.

1. Kid in King Arthur's Court, A (1995)
5
2. To Catch a Thief (1955)
4
3. Scream (1996)
3
4. Small Faces (1995)
2
5. So Dear to My Heart (1949)
1
6. Kim (1950)
3
7. Homeward Bound: The Incredible Journey (1993)
4
8. Mamma Roma (1962)
5
9. Wizard of Oz, The (1939)
1
10. Daniel Defoe's Robinson Crusoe (1996)
2
11. Braindead (1992)
3
12. Dragonheart (1996)
4
13. Sabrina (1954)
5
14. Ghosts of Mississippi (1996)
6
Input rating is not valid, ratings must be between 1 and 5 stars.
14. Ghosts of Mississippi (1996)
1
15. Cemetery Man (Dellamorte Dellamore) (1994)
2
16. Birdcage, The (1996)
3
17. Eye for an Eye (1996)
4
18. Stranger, The (1994)
5
19. Last Summer in the Hamptons (1995)
6
Input rating is not valid, ratings must be between 1 and 5 stars.
19. Last Summer in the Hamptons (1995)
4
20. S.F.W. (1994)
5
```

A continuación, tras un breve tiempo se muestra el vecindario como un diccionario con el identificador de un usuario y el grado de correlación con el usuario de la siguiente forma:

```
The actual neighborhood for user is (user:similarity):
 {396: 0.11617213184253895, 348: 0.10630316046259865, 141: 0.07742058010845326, 525: 0.07118806092964827, 67: 0.07046173881253959, 78: 0.06963213775588349, 434: 0.06915090567981634, 549: 0.06757927537542
298, 167: 0.06675503243806057, 45: 0.06482358389687332}
```

Y se muestran también las recomendaciones como un diccionario con el identificador de la película y la valoración estimada:



A continuación nos preguntará si deseamos ver las recomendaciones, presionando y nos mostrará la lista de la siguiente forma:

```
Do you want to see your recommendations? (y/n)
y
1. Babe (1995): 4.923188405797101
2. Much Ado About Nothing (1993): 4.923188405797101
3. Terminator 2: Judgment Day (1991): 4.923188405797101
4. Gone with the Wind (1939): 4.923188405797101
5. Big Night (1996): 4.923188405797101
6. Last of the Mohicans, The (1992): 4.923188405797101
7. Schindler's List (1993): 4.923188405797101
8. Muriel's Wedding (1994): 4.923188405797101
9. Butch Cassidy and the Sundance Kid (1969): 4.923188405797101
10. Jungle Book, The (1994): 4.923188405797101
11. Philadelphia Story, The (1940): 4.923188405797101
12. Wings of Desire (1987): 4.923188405797101
13. Deer Hunter, The (1978): 4.923188405797101
14. Man Who Would Be King, The (1975): 4.923188405797101
15. 39 Steps, The (1935): 4.923188405797101
16. Innocents, The (1961): 4.923188405797101
17. Old Man and the Sea, The (1958): 4.923188405797101
18. Delta of Venus (1994): 4.923188405797101
19. In & Out (1997): 4.919047619047619
20. Love! Valour! Compassion! (1997): 4.919047619047619
21. Smilla's Sense of Snow (1997): 4.870093457943925
22. Breakdown (1997): 4.870093457943925
23. Addicted to Love (1997): 4.870093457943925
24. Michael Collins (1996): 4.870093457943925
25. Contact (1997): 4.73489121506182
26. Mighty Aphrodite (1995): 4.695833333333333
27. Mystery Science Theater 3000: The Movie (1996): 4.695833333333333
28. Titanic (1997): 4.600091021047307
29. Mirror Has Two Faces, The (1996): 4.595454545454546
30. Marvin's Room (1996): 4.595454545454546
31. In Love and War (1996): 4.595454545454546
32. Family Thing, A (1996): 4.595454545454546
33. Boot, Das (1981): 4.58
34. Shawshank Redemption, The (1994): 4.533333333333333
35. Live Nude Girls (1995): 4.533333333333333
36. Sabrina (1995): 4.365609694895483
37. Star Wars (1977): 4.353844882881406
38. Star Trek: First Contact (1996): 4.3399660106429705
39. Tin Cup (1996): 4.334959027391139
40. Game, The (1997): 4.2835096022740995
41. Michael (1996): 4.268826782446315
42. Truth About Cats & Dogs, The (1996): 4.2074402789485195
43. Spitfire Grill, The (1996): 4.153180142605629
44. High School High (1996): 4.059407773273818
45. Craft, The (1996): 4.053274137017585
46. Independence Day (ID4) (1996): 4.043276561517846
47. Jack (1996): 4.021487732587219
48. Kiss the Girls (1997): 4.003703703703703
49. Othello (1995): 4.003703703703703
50. Inventing the Abbotts (1997): 4.003703703703703
```

Por último, nos preguntará de la misma forma si deseamos exportar las recomendaciones, presionando *y* nos notificará y se almacenará la lista de recomendaciones (por defecto en el fichero *u.recommendations*):

```
Do you want to export your recommendations? (y/n)
y


Recommendations exported successfully on ..ml-data/u.recommendations
```

Si abrimos dicho fichero, podremos ver la lista de recomendaciones:

```
1. Babe (1995): 4.923188405797101
2. Much Ado About Nothing (1993): 4.923188405797101
3. Terminator 2: Judgment Day (1991): 4.923188405797101
4. Gone with the Wind (1939): 4.923188405797101
5. Big Night (1996): 4.923188405797101
6. Last of the Mohicans, The (1992): 4.923188405797101
7. Schindler's List (1993): 4.923188405797101
8. Muriel's Wedding (1994): 4.923188405797101
9. Butch Cassidy and the Sundance Kid (1969): 4.923188405797101
10. Jungle Book, The (1994): 4.923188405797101
11. Philadelphia Story, The (1940): 4.923188405797101
12. Wings of Desire (1987): 4.923188405797101
13. Deer Hunter, The (1978): 4.923188405797101
14. Man Who Would Be King, The (1975): 4.923188405797101
15. 39 Steps, The (1935): 4.923188405797101
16. Innocents, The (1961): 4.923188405797101
17. Old Man and the Sea, The (1958): 4.923188405797101
18. Delta of Venus (1994): 4.923188405797101
19. In & Out (1997): 4.919047619047619
20. Love! Valour! Compassion! (1997): 4.919047619047619
21. Smilla's Sense of Snow (1997): 4.870093457943925
22. Breakdown (1997): 4.870093457943925
23. Addicted to Love (1997): 4.870093457943925
24. Michael Collins (1996): 4.870093457943925
25. Contact (1997): 4.73489121506182
26. Mighty Aphrodite (1995): 4.695833333333333
27. Mystery Science Theater 3000: The Movie (1996): 4.695833333333333
28. Titanic (1997): 4.600091021047307
29. Mirror Has Two Faces, The (1996): 4.595454545454546
30. Marvin's Room (1996): 4.595454545454546
31. In Love and War (1996): 4.595454545454546
32. Family Thing, A (1996): 4.595454545454546
33. Boot, Das (1981): 4.58
34. Shawshank Redemption, The (1994): 4.533333333333333
35. Live Nude Girls (1995): 4.533333333333333
36. Sabrina (1995): 4.365609694895483
37. Star Wars (1977): 4.353844882881406
38. Star Trek: First Contact (1996): 4.3399660106429705
39. Tin Cup (1996): 4.334959027391139
40. Game, The (1997): 4.2835096022740995
41. Michael (1996): 4.268826782446315
42. Truth About Cats & Dogs, The (1996): 4.2074402789485195
43. Spitfire Grill, The (1996): 4.153180142605629
44. High School High (1996): 4.059407773273818
45. Craft, The (1996): 4.053274137017585
46. Independence Day (ID4) (1996): 4.043276561517846
47. Jack (1996): 4.021487732587219
48. Kiss the Girls (1997): 4.003703703703703
49. Othello (1995): 4.003703703703703
50. Inventing the Abbotts (1997): 4.003703703703703
```

7. Referencias Bibliográficas

[1] Sistemas de recomendación | ¿Qué es el filtrado colaborativo?
Link Graph Everywhere.

[2] Recomendaciones basado en filtrado colaborativo en Python.
Link Stat Developer.

[3] Sistemas de recomendación en Python.
Link Findemor Blog.