Sistema de recomendación basado en filtrado colaborativo

José Alberto Gómez García

Índice de contenidos

Heramientas utilizadas 01

Arquitectura del software 02

Código desarrolado 03

Demostración 04

01

Herramientas empleadas

Herramientas empleadas

02

Arquitectura del software

Arquitectura del software

- CONFIGURACION
  - Dataclass con 12 variables.
- RECOMENDADOR
  - Constructor (pedir configuración es opcional)
  - GUI para pedir ficheros con las BDs (opcional)
  - Pedir valoraciones al usuario
  - Calcular vecindario
  - Calcular recomendaciones
  - Imprimir y/o exportar resultados
  - Método para ejecutar
- PROGRAMA PRINCIPAL

03

Código desarrolado

Configuración

```python
# Get the path of the project (parent of src)
BASE_PATH = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


@dataclass
class Configuration:
    DATA_DIR: str = os.path.join(BASE_PATH, 'data')
    RESULTS_DIR: str = os.path.join(BASE_PATH, 'results')
    MOVIES_FILE: str = os.path.join(DATA_DIR, 'u.item')
    RATINGS_FILE: str = os.path.join(DATA_DIR, 'u.data')
    RESULTS_FILE: str = os.path.join(RESULTS_DIR, 'recomendaciones.data')
    MOVIES_TO_BE_RATED: int = 20
    NEIGHBOURS: int = 10
    MAX_RESULTS: int = 30
    USE_MAX_RESULTS: bool = True
    DELETE_IRRELEVANT_USERS: bool = True
    THRESHOLD_TO_BE_IRRELEVANT: int = 3
    SHOW_DEBUG_INFO = True
```

Init del recomendador

```
self.peliculas = pd.read_csv(
    filepath_or_buffer=self.configuracion.MOVIES_FILE,
    sep='|',
    header=None,
    engine='python',
    encoding='mbcs',  # ANSI encoding in Windows, el fichero no está en UTF-8
    usecols=[0, 1, 2],
    names=['idMovie', 'title', 'date'])

self.valoraciones = pd.read_csv(
    filepath_or_buffer=self.configuracion.RATINGS_FILE,
    sep='\t',
    header=None,
    engine='python',
    encoding='mbcs',  # ANSI encoding in Windows, el fichero no está en UTF-8
    usecols=[0, 1, 2],
    names=['idUser', 'idMovie', 'rating'])

self.num_peliculas_a_valorar = self.configuracion.MOVIES_TO_BE_RATED
self.tamanio_vecindario = self.configuracion.NEIGHBOURS
self.vecindario = {}
self.valoraciones_por_usuario = {}
self.recomendaciones = {}
```

Obviamos la petición de parámetros al usuario.

Pedir ficheros

```python
@staticmethod
def pedir_fichero(title, initialdir=configuracion.DATA_DIR, filetypes=None):
    if filetypes is None:
        filetypes = [('Todos los ficheros', '.*')]
    root = Tk()
    root.withdraw()  # Ocultar ventana padre de Tk y mostrar la nuestra como popup
    root.attributes('-topmost', True)
    try:
        fichero = filedialog.askopenfilename(
            initialdir=initialdir, title=title,
            filetypes=filetypes)
        root.destroy()
    except (OSError, FileNotFoundError):
        print('No se ha podido abrir el fichero.')
        sys.exit(1)
    except Exception as e:
        print(f'Ha ocurrido un error inesperado: {e}')
        sys.exit(1)
    if len(fichero) == 0 or fichero is None:
        print('No se ha seleccionado ningún fichero.')
        sys.exit(1)

    return fichero
```

Se muestra un menú gráfico para hacer la selección.

Pedir valoraciones

```python
def pedir_valoraciones_al_usuario(self):
    peliculas_a_valorar = self.peliculas.sample(n=self.num_peliculas_a_valorar, replace=False)
    print('Por favor, valora las siguientes películas del 1 al 5:')

    contador = 1
    for _, row in peliculas_a_valorar.iterrows():
        titulo = row['title']
        while True:
            print(f'Película {contador} de {self.num_peliculas_a_valorar}: {titulo}')
            valoracion = input('Valoración: ')
            if valoracion == "" or int(valoracion) not in range(1, 6):
                print('Valoración no válida. Estas deben estar entre 1 y 5 estrellas.')
            else:
                break
        self.valoraciones_por_usuario[row['idMovie']] = int(valoracion)
        contador += 1

    print('Gracias por tu colaboración al valorar las películas. \n')
```

Calcular vecindario I

```python
def calcular_vecindario(self):
    print('Calculando vecindario...\n')

    usuarios = self.valoraciones['idUser'].unique()
    usuarios = list(np.insert(usuarios, 0, 0))  # Mi usuario es el 0, la DB empieza por 1.

    # Crear matriz que agrupe los usuarios y las películas
    matriz_valoraciones = pd.DataFrame(index=usuarios, columns=self.peliculas['idMovie'])
    matriz_valoraciones = matriz_valoraciones.fillna(0)

    # Rellenar la matriz con las valoraciones del usuario
    for movie in self.valoraciones_por_usuario.keys():
        matriz_valoraciones.iloc[0, movie - 1] = self.valoraciones_por_usuario[movie]
    # Añadir las valoraciones de los demás usuarios
    for usuario in usuarios[1:]:
        valoraciones_usuario = self.valoraciones[self.valoraciones['idUser'] == usuario]
        for _, row in valoraciones_usuario.iterrows():
            matriz_valoraciones.iloc[usuario, row['idMovie'] - 1] = row['rating']
```

Calcular vecindario II

```
# Elimino a los usuarios con menos de X coincidencias, quedandome así con los de más de X coincidencias.
if self.configuracion.DELETE_IRRELEVANT_USERS:
    usuarios_a_eliminar = list()
    grupos = self.valoraciones.groupby('idUser')['idMovie'].apply(list)
    for usuario, peliculas in grupos.items():
        common = set(self.valoraciones_por_usuario.keys()).intersection(set(peliculas))
        if len(common) < self.configuracion.THRESHOLD_TO_BE_IRRELEVANT:
            usuarios_a_eliminar.append(usuario)
    matriz_valoraciones.drop(matriz_valoraciones[usuarios_a_eliminar], axis=0, inplace=True)
    # Obtener el numero real de filas, al eliminar usuarios puede ser menor que el tamaño del vecindario
    num_filas = matriz_valoraciones.shape[0]
    if num_filas < self.tamanio_vecindario:
        self.tamanio_vecindario = num_filas - 1

correlation_matrix = np.corrcoef(matriz_valoraciones)
usuarios_ordenados = np.argsort(correlation_matrix[0])[::-1]

for i in range(1, self.tamanio_vecindario + 1):
    self.vecindario[usuarios_ordenados[i]] = correlation_matrix[0][usuarios_ordenados[i]]

if self.configuracion.SHOW_DEBUG_INFO:
    print('Vecindario (idUser : correlation):\n', self.vecindario)
```

Recomendaciones I

$$\hat{r}(u,i) = \bar{r}(u) + C \sum_{v \in N_u^k} sim(u,v)\left(r(v,i) - \bar{r}(v)\right)$$

$$C = \frac{1}{\sum_{v \in N_u^k} |sim(u,v)|}$$

Diapositiva 28 del tema 2.

Recomendaciones II

```python
def calcular_recomendaciones(self):
    print('Calculando recomendaciones...\n')
    peliculas_valoradas = self.valoraciones_por_usuario.keys()
    valoraciones_medias = np.mean(list(self.valoraciones_por_usuario.values()))
    vecindario_media = {}
    for usuario in self.vecindario.keys():
        valoracion_vecindario = self.valoraciones[self.valoraciones['idUser'] == usuario]
        media_vecindario = np.mean(valoracion_vecindario['rating'])
        vecindario_media[usuario] = media_vecindario
    # Para cada pelicula no vista por el usuario, calcular el posible interés.
    for pelicula in self.peliculas['idMovie']:
        if pelicula not in peliculas_valoradas:
            similitud, corr_acum = 0, 0
            for usuario in self.vecindario.keys():
                correlacion = self.vecindario[usuario]
                valoracion = self.valoraciones[(self.valoraciones['idUser'] == usuario)
                                               & (self.valoraciones['idMovie'] == pelicula)]['rating']
                if len(valoracion) > 0:
                    similitud += correlacion * (valoracion.values[0] - vecindario_media[usuario])
                    corr_acum += abs(correlacion)
            interes = 0
            if corr_acum > 0:
                interes = (similitud / corr_acum) + valoraciones_medias
                interes = min(5, interes)  # El interés no puede ser mayor que 5 estrellas
            if interes >= 4:
                self.recomendaciones[pelicula] = interes
    # Ordenar las recomendaciones ordenadas por interés descendente
    self.recomendaciones = dict(sorted(self.recomendaciones.items(), key=lambda item: item[1], reverse=True))
```

Imprimir y exportar

```python
def imprimir_recomendaciones(self):
    contador = 1
    for pelicula in self.recomendaciones.keys():
        if self.configuracion.USE_MAX_RESULTS and contador > self.configuracion.MAX_RESULTS:
            break
        titulo = self.peliculas[self.peliculas['idMovie'] == pelicula]['title'].values[0]
        print(f'{contador}. {titulo}. \tPuntuacion: {self.recomendaciones[pelicula]:.3f}')
        contador += 1

def exportar_recomendaciones(self):
    contador = 1
    directorio = self.configuracion.RESULTS_DIR
    if not os.path.exists(directorio):
        os.makedirs(directorio)

    with open(self.configuracion.RESULTS_FILE, 'x') as f:
        for pelicula in self.recomendaciones.keys():
            titulo = self.peliculas[self.peliculas['idMovie'] == pelicula]['title'].values[0]
            f.write(f'{contador}. {titulo}. \tPuntuación: {self.recomendaciones[pelicula]:.3f}\n')
            contador += 1
```

Ejecutar

```
def ejecutar(self):
    self.pedir_valoraciones_al_usuario()
    self.calcular_vecindario()
    self.calcular_recomendaciones()
    while True:
        imprimir = input('¿Desea imprimir las recomendaciones? (S/N): ')
        imprimir.upper()
        if imprimir == 'S' or imprimir == 'N':
            break
        else:
            print('Opción no válida. Introduzca S o N.')
    if imprimir == 'S':
        self.imprimir_recomendaciones()
    while True:
        exportar = input('¿Desea exportar las recomendaciones? (S/N): ')
        exportar.upper()
        if exportar == 'S' or exportar == 'N':
            break
        else:
            print('Opción no válida. Introduzca S o N.')
    if exportar == 'S':
        try:
            self.exportar_recomendaciones()
            print('Recomendaciones exportadas correctamente.')
        except FileExistsError:
            print('Error al exportar las recomendaciones.')
    print('Fin del programa. Gracias por usar el sistema de recomendación.')
```

Programa principal

```python
from src.recomendador import Recomendador

if __name__ == '__main__':
    print('Bienvenido al recomendador de películas.\n')

    recomendador = Recomendador()
    recomendador.ejecutar()
```

04

Demostración

Configuración

```
Bienvenido al recomendador de películas.

¿Desea utilizar la configuración por defecto? (S/N): N
Introduzca el número de películas a valorar: 20
Introduzca el tamaño del vecindario: 10
¿Desea limitar el número de recomendaciones? (S/N): S
Introduzca el número máximo de recomendaciones: 10
¿Desea eliminar usuarios irrelevantes? (S/N): S
Introduzca el número mínimo de películas en común con nosotros para considerar al usuario relevante: 3
```

La ruta a ficheros se pide por GUI.
No todos los parámetros se pueden cambiar aquí.

Pedir valoraciones

```
Por favor, valora las siguientes películas del 1 al 5:
Película 1 de 20: Virtuosity (1995)
Valoración: 3
Película 2 de 20: New York Cop (1996)
Valoración: 4
Película 3 de 20: Amos & Andrew (1993)
Valoración: 2
Película 4 de 20: One Flew Over the Cuckoo's Nest (1975)
Valoración: 4
Película 5 de 20: Grace of My Heart (1996)
Valoración: 5
Película 6 de 20: 3 Ninjas: High Noon At Mega Mountain (1998)
Valoración: 3
Película 7 de 20: Candidate, The (1972)
Valoración: 2
Película 8 de 20: Nell (1994)
Valoración: 4
Película 9 de 20: Mighty, The (1998)
Valoración: 5
Película 10 de 20: Pollyanna (1960)
Valoración: 3
```

```
Película 11 de 20: Spice World (1997)
Valoración: 4
Película 12 de 20: When Harry Met Sally... (1989)
Valoración: 5
Película 13 de 20: Dead Presidents (1995)
Valoración: 3
Película 14 de 20: Liar Liar (1997)
Valoración: 4
Película 15 de 20: Wooden Man's Bride, The (Wu Kui) (1994)
Valoración: 2
Película 16 de 20: Paris, France (1993)
Valoración: 3
Película 17 de 20: Stranger in the House (1997)
Valoración: 4
Película 18 de 20: It Happened One Night (1934)
Valoración: 5
Película 19 de 20: Panther (1995)
Valoración: 3
Película 20 de 20: Charade (1963)
Valoración: 2
Gracias por tu colaboración al valorar las películas.
```

Resultados I

```
Calculando vecindario...

Vecindario (idUser : correlation):
 {183: 0.1184948105318245, 15: 0.11097747380579606, 56: 0.10551335414456821, 11: 0.10361335385783499, 142: 0.09070835213228645,
Calculando recomendaciones...

¿Desea imprimir las recomendaciones? (S/N): S
1. Good, The Bad and The Ugly, The (1966).  Puntuacion: 5.000
2. Restoration (1995).  Puntuacion: 5.000
3. Rosewood (1997).     Puntuacion: 5.000
4. Mrs. Brown (Her Majesty, Mrs. Brown) (1997).     Puntuacion: 5.000
5. FairyTale: A True Story (1997).  Puntuacion: 5.000
6. Kiss the Girls (1997).   Puntuacion: 5.000
7. Day the Earth Stood Still, The (1951).   Puntuacion: 5.000
8. Nixon (1995).    Puntuacion: 5.000
9. Cry, the Beloved Country (1995).     Puntuacion: 5.000
10. My Fair Lady (1964).    Puntuacion: 5.000
¿Desea exportar las recomendaciones? (S/N): S
Recomendaciones exportadas correctamente.
Fin del programa. Gracias por usar el sistema de recomendación.
```

Resultados por terminal.

Resultados II

```
90. Remains of the Day, The (1993).       Puntuación: 4.434
91. Young Frankenstein (1974).      Puntuación: 4.434
92. Courage Under Fire (1996).      Puntuación: 4.429
93. Chasing Amy (1997).       Puntuación: 4.411
94. Apollo 13 (1995).   Puntuación: 4.399
95. Sabrina (1995).     Puntuación: 4.375
96. Swingers (1996).    Puntuación: 4.372
97. Ghost (1990).       Puntuación: 4.367
98. Pulp Fiction (1994).      Puntuación: 4.350
99. Wrong Trousers, The (1993).     Puntuación: 4.345
100. In & Out (1997).   Puntuación: 4.341
101. Seven Years in Tibet (1997).   Puntuación: 4.341
102. Michael Collins (1996).  Puntuación: 4.341
103. Dolores Claiborne (1994).      Puntuación: 4.335
104. Searching for Bobby Fischer (1993).  Puntuación: 4.335
105. Wallace & Gromit: The Best of Aardman Animation (1996).      Puntuación: 4.335
```

Resultados del fichero exportado.

Gracias

¿Alguna pregunta o comentario?

José Alberto Gómez García
modej@correo.ugr.es