



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

Gestión del Conocimiento en las Organizaciones:

Sistemas de recomendación

Andrea Calero Caro
(alu0101202952@ull.edu.es)



Índice:

1. Introducción.	2
2. Desarrollo y explicación del funcionamiento.	2
2.1. CoeficientePearson(usux, usuy).	8
2.2. DCoseno(usux, usuy).	12
2.3. DEuclidea(usux, usuy).	14
2.4. PrediccionSimple(simi[pearson coseno euclidea], vecinos, usuy).	16
2.4. PrediccionDMedia(simi[pearson coseno euclidea], vecinos, usux, usuy).	17
3. Funcionamiento.	18
4. Conflictos.	20
5. Referencias.	20



1. Introducción.

En ésta práctica se quiere implementar un sistema de recomendación, el cuál maneja las métricas de: **Correlación de Pearson**, **Distancia coseno** y **Distancia Euclídea**, luego se le pasará un **mínimo de 3 vecinos** y el tipo de predicción: **Simple** o **Distancia con la Media**.

Para ejecutar el programa, en python, será:

```
> python [program.py] [matriz.txt]
```

El programa en python se llama: **sistema-recomenda.py** y luego se pasará un archivo .txt con la matriz de utilidad que tendrá las valoraciones de cada usuario, este con el formato que se ha explicado en clase.

2. Desarrollo y explicación del funcionamiento.

Se procederá a explicar el código en orden a su funcionamiento y acceso a cada método del programa.

Como se comentó anteriormente, se trabajará con una métrica, unos vecinos y un tipo de predicción, sin embargo, estos no se pasan por argumento, sino por un menú, esto debido a que es más visual para poder trabajar con los datos.

Como paso previo se creó un método que pide un número entero y comprueba que sea entero, en otro caso mostrará un fallo.

```
246 #####
247
248 # Pedir numero, control de errores al recibir
249 def pedirNumeroEntero():
250     correcto = False
251     num = 0
252     while(not correcto):
253         try:
254             num = int(input("Introduce un numero entero: "))
255             correcto=True
256         except ValueError:
257             print('Error, introduce un numero entero')
258     return num
259 salir = False
260 opcion = 0
261
262 #####
263
```



También, en las primeras líneas del programa se comprueba que el paso de argumentos por línea de comando, sea el correcto, con la librería **sys**. Ésta en sustitución de la librería **argparse**.

```
9
10 # Importar librerías
11 import sys
12 import math
13 import numpy as np
14
15 # Comprobar numero de argumentos
16 nvar=1
17 if len(sys.argv) != nvar+1:
18     sys.exit('El numero de argumentos no es el correcto ('+str(nvar)+')')
19 p=[(i) for i in sys.argv[1:nvar+1]]
20
21
22 #####
23
```

El menú de opciones consta de 3 partes, el que recoge el tipo de métrica:

- 1) Correlación de Pearson
- 2) Distancia coseno
- 3) Distancia Euclídea

Por otro lado recogerá el número de vecinos, el cuál mínimo tiene que haber 3. Y finalmente, se recoge el tipo de Predicción:

- 1) Predicción Simple
- 2) Predicción Distancia con la Media.

Una vez elegida la métrica, se pasa a procesar el archivo de texto que tiene a la matriz por la función **LeerFichero()**. Y la cuál devolverá dos usuarios, usux e usuy, que corresponde con los dos que se analizarán. Continúa así, con la llamada a la función correspondiente a la métrica deseada: **CoeficientePearson(usux, usuy)**, **DCoseno(usux, usuy)** o **DEuclídea(usux, usuy)**. Sacando finalmente en dicho apartado el resultado de la métrica y un mensaje si la similitud cumple con lo establecido, para ello se llaman a las funciones: **SimilitudPearson(resultadopearson)**, **SimilitudCoseno(resultadocoseno)**.

Mostrándose así el código:



```
264 # Menu de opciones
265 while not salir:
266     print("OPCIONES DISPONIBLES")
267     print ("1. Correlacion de Pearson")
268     print ("2. Distancia coseno")
269     print ("3. Distancia Euclidea")
270     print ("4. Salir")
271
272     print ("Elige una opcion")
273     opcion = pedirNumeroEntero()
274     print("#####")
275     if opcion == 1:
276         print ("\nOpcion 1: 'Correlacion de Pearson'\n")
277
278         # Matriz usuarios/recomendacion
279         matriz = sys.argv[1]
280         usux, usuy = LeerFichero()
281         resultadopearson = CoeficientePearson(usux, usuy)
282         print("\n")
283         print("Coeficiente de Pearson: ", resultadopearson)
284         print("\n")
285         print("Similitud: ", SimilitudPearson(resultadopearson))
286         print("\n")
287
288     elif opcion == 2:
289         print ("\nOpcion 2: 'Distancia coseno'\n")
290
291         # Matriz usuarios/recomendacion
292         matriz = sys.argv[1]
293         usux, usuy = LeerFichero()
294         resultadocoseno = DCoseno(usux, usuy)
295         print("\n")
296         print("Distancia Coseno: ", resultadocoseno)
297         print("\n")
298         print("Similitud: ", SimilitudCoseno(resultadocoseno))
299         print("\n")
300
301     elif opcion == 3:
302         print ("\nOpcion 3: 'Distancia Euclidea'\n")
303
304         # Matriz usuarios/recomendacion
305         matriz = sys.argv[1]
306         usux, usuy = LeerFichero()
307         resultadoeuclidea = DEuclidea(usux, usuy)
308         print("\n")
309         print("Distancia Euclidea: ", resultadoeuclidea)
310         print("\n")
311
312     elif opcion == 4:
313         salir = True
314     else:
315         print ("Introduce un numero entre 1 y 3")
316         print("#####")
317
318     while True:
319         #numvecinos = 0
320         print("#INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS")
```



La siguiente parte del menú consta en recoger número de vecinos y tipo de predicción.

El tipo de predicción será 1 para Predicción Simple y 2 para la Distancia con la Media, cada opción del menú tendrá dos funciones asociadas: **PrediccionSimple(simi[pearson | coseno | euclidea], vecinos, usuy)** y **PrediccionDMedia(simi[pearson | coseno | euclidea], vecinos, usux, usuy)**

Por un lado la predicción Simple requiere del resultado de similitud que se obtiene en cada métrica, los vecinos y el usuario y, o v en caso de las diapositivas. Mientras que, por otro lado, la predicción distancia con la media, requiere además de estos el usuario x, o u, en caso de las diapositivas. Más adelante, en la explicación de ambas fórmulas se extenderá.



```
318 while True:
319     #numvecinos = 0
320     print("INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS")
321     vecinos = pedirNumeroEntero()
322     if vecinos >= 3:
323         break
324     print("Introduce mas de 2 vecinos, minimo 3\n")
325
326
327 while True:
328     print("INTRODUCE EL TIPO DE PREDICCION")
329     print ("1. Prediccion simple")
330     print ("2. Diferencia con la media")
331     prediccion = pedirNumeroEntero()
332     if prediccion == 1:
333         print ("\nOpcion 1: 'Prediccion simple'\n")
334         if opcion == 1: #Correlacion de Pearson
335             simipearson = CoeficientePearson(usux, usuy)
336             prediccionspearson = PrediccionSimple(simipearson, vecinos, usuy)
337             print("\n")
338             print("Prediccion: ", prediccionspearson)
339         elif opcion == 2: #Distancia coseno
340             simicoseno = DCoseno(usux, usuy)
341             prediccionscoseno = PrediccionSimple(simicoseno, vecinos, usuy)
342             print("\n")
343             print("Prediccion: ", prediccionscoseno)
344         elif opcion == 3: #Distancia euclidea
345             simieuclicea = DEuclidean(usux, usuy)
346             prediccioneuclidean = PrediccionSimple(simieuclicea, vecinos, usuy)
347             print("\n")
348             print("Prediccion: ", prediccioneuclidean)
349         else:
350             print("Error con la prediccion simple")
351     elif prediccion == 2:
352         print ("\nOpcion 2: 'Diferencia con la media'\n")
353         if opcion == 1: #Correlacion de Pearson
354             simipearson = CoeficientePearson(usux, usuy)
355             prediccioneuclidean = PrediccionDMedia(simipearson, vecinos, usux, usuy)
356             print("\n")
357             print("Prediccion: ", prediccioneuclidean)
358         elif opcion == 2: #Distancia coseno
359             simicoseno = DCoseno(usux, usuy)
360             prediccioneuclidean = PrediccionDMedia(simicoseno, vecinos, usux, usuy)
361             print("\n")
362             print("Prediccion: ", prediccioneuclidean)
363         elif opcion == 3: #Distancia euclidea
364             simieuclicea = DEuclidean(usux, usuy)
365             prediccioneuclidean = PrediccionDMedia(simieuclicea, vecinos, usux, usuy)
366             print("\n")
367             print("Prediccion: ", prediccioneuclidean)
368         else:
369             print("Error con la prediccion diferencia con la media")
370     else:
371         print("#####")
372         print ("Introduce un numero, 1 o 2")
373         print("#####")
374     print ("Fin")
```



Ahora se profundizará, en orden de acceso, a las funciones que se definieron anteriormente.

Primero, si se introduce la métrica se pasará a llamar a la función **LeerFichero()**, la función no recibe nada, y devuelve los dos usuarios en cada iteración del recorrido de la matriz. Para ello se abre el archivo .txt y se lee línea a línea guardando como usux la línea de la iteración **i** de la matriz y el usuy la línea de iteración **i+1**. Así tendríamos los dos usuarios. Y en la variable **datosusuario = []** se guardará la matriz original de utilidad.

```
34 #####
35
36 # Tratamiento de argumentos
37 # Leer línea a línea, guardando valoracion usuario
38 # matrix = sys.argv[1]
39 def LeerFichero():
40     print ("+++ Matriz relacion usux/usuy de los usuarios +++")
41     datosusuario = []
42     usux = []
43     usuy = []
44     with open("matriz.txt") as fname:
45         usuarios = fname.readlines()
46         for i in range(len(usuarios)):
47             if i+1 < len(usuarios):
48                 usux.append(usuarios[i])
49                 usuy.append(usuarios[i+1])
50                 # Se llama al metodo para limpiar blancos y espacios
51                 newusux, newusuy = LimpiarEspacios(usux, usuy)
52                 print(newusux, newusuy)
53                 CoeficientePearson(newusux, newusuy)
54                 #print ("Usuario " + i + "Valoracion: " + usuarios[i])
55             datosusuario.append(usuarios)
56         print("\n")
57         print("Matriz original de utilidad: ", datosusuario)
58         print("usux: ", usux)
59         print("usuy: ", usuy)
60         return usux, usuy
61     #usux[:] = []
62     #usuy[:] = []
63     print("\n")
64
65
66 #####
67
```

Como se observaba que habían espacios en blanco y elementos extras que ensuciaba el código se pasó a implementar una función **LimpiarEspacios(usux, usuy)**, ésta se tuvo que implementar para no hacer un código extenso, seguir un modelo de encapsulamiento y por errores con el método ya implementado **.strip()**

Quedando ésta función tal que:



```
22 #####
23
24 # Limpiar espacios en blanco
25 def LimpiarEspacios(usux, usuy):
26     for i in range(len(usux)):
27         if i < len(usux):
28             newusux = usux[i].strip()
29             newusuy = usuy[i].strip()
30     return newusux, newusuy
31     #print ("Limpio: ", newusux, newusuy)
32
33
34 #####
35
```

Luego de que se lea el fichero y limpie espacios en blanco o saltos de línea, lo siguiente es en cada opción, Pearson, Coseno o Euclídea, implementar la función correspondiente, siguiendo las fórmulas matemáticas y explicaciones dadas en clase.

2.1. CoeficientePearson(usux, usuy).

Esta función se basó en la fórmula matemática:

Medidas de similitud: Correlación de Pearson

Índice que puede utilizarse para medir el **grado de relación de dos variables** siempre y cuando ambas sean cuantitativas y continuas.

- u, v usuarios
- $r(u, i)$ calificación del usuario u del ítem i
- $\bar{r}(u)$ media de calificaciones del usuario u
- S_{uv} conjunto de ítems calificados por u y v , $S_u = \{i \in I / r(u, i) \neq \emptyset\}$ conjunto de ítems calificados por u y $S_{uv} = S_u \cap S_v$

$$sim(u, v) = \frac{\sum_{i \in S_{uv}} (r(u, i) - \bar{r}(u)) \cdot (r(v, i) - \bar{r}(v))}{\sqrt{\sum_{i \in S_{uv}} (r(u, i) - \bar{r}(u))^2} \sqrt{\sum_{i \in S_{uv}} (r(v, i) - \bar{r}(v))^2}}$$



Las variables que se usaron fueron:

- **vcalificacionx** e **vcalificaciony**, que corresponden a el resultado de las calificaciones que dieron los usuarios x e y, respectivamente. Las cuales se implementaron en una función **CalificacionObjeto([usuariox | usuarioy])**.
- **mediax** y **mediay**, que corresponden a el resultado de las medias devueltas en el par de resultado que retorna la función **Media(vcalificacionx, vcalificaciony)**
- **numerador** y **denominador**, que contendrán el numerador y denominador con las operaciones que se realizan siguiendo la fórmula planteada.
- **xpow** e **ypow**, estos corresponden a la potencia con la que se trabajará. Todos como tipo **float**.
- **pearson[]**, es la lista que tendrá el resultado de la similitud que nos pide la fórmula.

Para hacer las operaciones se requirió de la librería **math**, ya que tiene métodos ya implementados de la potencia **.pow()** a la que se le pasa la variable en este caso la resta de la calificación del usuario y su media, y se elevaría a 2, como se indica. Luego el denominador es un producto de raíces que se representa con **.sqrt(xpow*ypow)**, xpow e ypow contienen las potencias que se nombró. Finalmente, Pearson recogería el numerador/denominador.

Lo tenía implementado de otra manera pero me daba error, el error que se muestra en los comentarios de la siguiente imagen:

```
113 #####
114
115 def CoeficientePearson(usuariox, usuarioy):
116     pearson = []
117     # Calificaciones del usuario sobre objeto i
118     vcalificacionx = CalificacionObjeto(usuariox)
119     vcalificaciony = CalificacionObjeto(usuarioy)
120
121     # Media de calificaciones de los usuarios
122     mediax, mediay = Media(vcalificacionx, vcalificaciony)
123     #print("La media usuariox = ", mediax)
124     #print("La media usuarioy = ", mediay)
125     #print("\n")
126
127     numerador = 0.0
128     xpow = 0.0
129     ypow = 0.0
130     for i in range(len(usuariox)):
131         #numerador += vcalificacionx[i]-mediax*(vcalificaciony[i]-mediay) //unsupported operand type(s) for -: 'str' and 'float'
132         numerador += i-mediay
133     for i in range(len(usuariox)):
134         #xpow = xpow + (math.pow(vcalificacionx[i]-mediax,2)) //unsupported operand type(s) for -: 'str' and 'float'
135         xpow += math.pow(i-mediay,2)
136     for i in range(len(usuariox)):
137         #ypow = ypow + (math.pow(vcalificaciony[i]-mediay,2)) //unsupported operand type(s) for -: 'str' and 'float'
138         ypow += math.pow(i-mediay,2)
139     # Producto de las raices cuadradas de los usuarios x e y
140     denominador = math.sqrt(xpow*ypow)
141     pearson = numerador/denominador
142     return pearson
143
144
145 #####
```

La función **CalificacionObjeto()** sería:



```
66 #####
67
68 # Calificaciones del usuario sobre el objeto
69 def CalificacionObjeto(usuario):
70     resultado = []
71     for i in range(len(usuario)):
72         if i != "-" and i != " ":
73             resultado.append(usuario[i])
74     return resultado
75
76
77 #####
```

Sólo se encarga de mostrar cada calificación del objeto de un usuario.

Y la función **Media()** sería:

```
88 #####
89
90 # Media
91 def Media(usux, usuy):
92     sumax = 0
93     sumay = 0
94     lenvector = 0
95     for i in range(len(usux)):
96         for j in range(len(usuy)):
97             if i < len(usux) and j < len(usuy):
98                 aux=int(i)
99                 aux2=int(j)
100                 sumax += aux
101                 sumay += aux2
102                 lenvector = lenvector+1
103             if lenvector==0 or i == "-" or j == "-" or i == " " or j == " ":
104                 return 0,0
105             else:
106                 xresult = float(sumax)/lenvector
107                 yresult = float(sumay)/lenvector
108                 i+1,j+1
109     return xresult, yresult
110
111
112
113 #####
```



Esta función, se encarga de sumar cada valor de la lista y dividirlo entre el total, teniendo en cuenta que como excepción se puede encontrar un carácter “-” o “ ”, pudiendo el código poder definirlos como un cero.

A su vez, cuando se calcula este coeficiente de similitud, se procede a una función de **SimilitudPearson(resultadopearson)**, ésta tiene mensajes para corroborar que el resultado está dentro de unos parámetros establecidos.

```
216 #####
217
218 def SimilitudPearson(valor):
219     if valor == 1:
220         print("Correlacion directa perfecta")
221     elif 0 < valor < 1:
222         print("Correlacion directa")
223     elif valor == 0:
224         print("No hay correlacion")
225     elif -0 < valor < 0:
226         print("Correlacion inversa")
227     elif valor == -1:
228         print("Correlacion inversa perfecta")
229     else:
230         print("No se encuentra similitud")
231
232
233 #####
234
```

Y la cual se basó en la fórmula:

Medidas de similitud: Correlación de Pearson

Los valores posibles de similitud van desde -1 hasta 1 :

- Si $\text{sim}(u, v) = 1$, correlación directa perfecta
- Si $0 < \text{sim}(u, v) < 1$, correlación directa
- Si $\text{sim}(u, v) = 0$, no hay correlación
- Si $-1 < \text{sim}(u, v) < 0$, correlación inversa
- Si $\text{sim}(u, v) = -1$, correlación inversa perfecta



2.2. DCoseno(usux, usuy).

Esta función se basó en la fórmula matemática:

Medidas de similitud: Distancia Coseno

Si dos vectores tienen exactamente la misma orientación (el ángulo que forman es 0°) su coseno toma el valor de **1**, si son **perpendiculares** (forman un ángulo de 90°) su coseno es **0** y si tienen **orientaciones opuestas** (ángulo de 180°) su coseno es de **-1**

$$sim(u, v) = \frac{\sum_{i \in S_{uv}} r(u, i) \cdot r(v, i)}{\sqrt{\sum_{i \in S_{uv}} (r(u, i))^2} \sqrt{\sum_{i \in S_{uv}} (r(v, i))^2}}$$

Los valores posibles de similitud van desde 0 hasta 1:

- Si $sim(u, v) = 1$, correlación directa perfecta.
- Si $0 < sim(u, v) < 1$, correlación directa.
- Si $sim(u, v) = 0$, no hay correlación.

Las variables que se usaron fueron:

- **vcalificacionx** e **vcalificaciony**, que corresponden a el resultado de las calificaciones que dieron los usuarios x e y, respectivamente. Las cuales se implementaron en una función **CalificacionObjeto([usuariox | usuarioy])**. Ya explicado anteriormente.
- **numerador** y **denominador**, que contendrán el numerador y denominador con las operaciones que se realizan siguiendo la fórmula planteada.
- **xpow** e **ypow**, estos corresponden a la potencia con la que se trabajará. Todos como tipo **float**.
- **coseno[]**, es la lista que tendrá el resultado de la similitud que nos pide la fórmula.

Si se sigue la fórmula se guardaría en las variables xpow e ypow, las potencias elevadas a 2 de cada elemento de la lista. Luego, en el denominador se realizaría un producto de raíces, como también se vio antes.

Finalmente, la lista coseno[] tiene el resultado de la similitud que sale del numerador / denominador.

Quedando el código tal que:



```
145 #####
146
147 def DCoseno(usuariox, usuarioy):
148     coseno = []
149     # Calificaciones del usuario sobre objeto i
150     vcalificacionx = CalificacionObjeto(usuariox)
151     vcalificaciony = CalificacionObjeto(usuarioy)
152
153     numerador = 0.0
154     xpow = 0.0
155     ypow = 0.0
156     for i in range(len(vcalificacionx)):
157         for j in range(len(vcalificaciony)):
158             numerador = i*j
159             xpow += math.pow(i,2)
160             ypow += math.pow(j,2)
161             denominador = math.sqrt(xpow*ypow)
162         coseno = numerador/denominador
163     return coseno
164
165 #####
166
```

A su vez, cuando se calcula este coeficiente de similitud, se procede a una función de **SimilitudCoseno(resultado coseno)**, ésta tiene mensajes para corroborar que el resultado está dentro de unos parámetros establecidos.

```
233 #####
234
235 def SimilitudCoseno(valor):
236     if valor == 1:
237         print("Correlacion directa perfecta")
238     elif 0 < valor < 1:
239         print("Correlacion directa")
240     elif valor == 0:
241         print("No hay correlacion")
242     else:
243         print("No se encuentra similitud")
244
245 #####
246
```



Y la cual se basó en la fórmula:

Los valores posibles de similitud van desde 0 hasta 1:

- Si $\text{sim}(u, v) = 1$, correlación directa perfecta.
- Si $0 < \text{sim}(u, v) < 1$, correlación directa.
- Si $\text{sim}(u, v) = 0$, no hay correlación.

2.3. DEuclidean(usux, usuy).

Esta función se basó en la fórmula matemática:

Medidas de similitud: Distancia Euclídea

Entre dos puntos p y q se define como **la longitud del segmento que une ambos puntos**. Puede generalizarse para un **espacio Euclídeo n -dimensional**.

- u, v : usuarios
- $r(v, i)$: clasificación del usuario v para el ítem i
- P : conjunto de ítems clasificados por u y v

$$d(u, v)_{euc} = \sqrt{\sum_{p \in P} (r(u, i) - r(v, i))^2}$$

Las variables que se usaron fueron:

- **vcalificacionx** e **vcalificaciony**, que corresponden a el resultado de las calificaciones que dieron los usuarios x e y , respectivamente. Las cuales se implementaron en una función **CalificacionObjeto([usuariox | usuarioy])**. Ya explicado anteriormente.
- **vcalificacionxy**, que corresponden al resultado de las calificaciones que dieron en común los dos usuarios, x e y . Las cuales se implementaron en una función **CalificacionIgualObjeto(vcalificacionx, vcalificaciony)**
- **xypow**, esto corresponde a la potencia con la que se trabajará. Todos como tipo **float**.
- **euclidean[]**, es la lista que tendrá el resultado de la similitud que nos pide la fórmula.



Si se sigue la fórmula se guardaría en la variable `xypow`, la potencia elevadas a 2 de la resta de los elementos en común de los elementos de la lista. Luego, se realizaría la raíz de dicha potencia.

Finalmente, la lista `euclidea[]` tiene el resultado de la similitud que sale de la anterior operación.

Quedando el código tal que:

```
166 #####
167
168 def DEuclidea(usuariox, usuarioy):
169     deuclydea = []
170     # Calificaciones del usuario sobre objeto i
171     vcalificacionx = CalificacionObjeto(usuariox)
172     vcalificaciony = CalificacionObjeto(usuarioy)
173
174     # Misma calificaciones de ambos usuarios sobre un objeto i
175     # Duda sobre como implementarlo
176     vcalificacionxy = CalificacionIgualObjeto(vcalificacionx, vcalificaciony)
177
178     xypow = 0.0
179     for i in range(len(vcalificacionx)):
180         for j in range(len(vcalificaciony)):
181             xypow = math.pow(i-j,2)
182     deuclydea = math.sqrt(xypow)
183     return deuclydea
184
185
186 #####
```

La función **CalificacionIgualObjeto(vcalificacionx, vcalificaciony)**, tiene como objetivo encontrar aquellas calificaciones de la lista que es común en valoración, con respecto a ambos usuarios. Tal que:

```
77 #####
78
79 # Misma Calificaciones de los dos usuarios sobre un objeto
80 def CalificacionIgualObjeto(usuariox, usuarioy):
81     resultado = []
82     for i in range(len(usuariox)):
83         for j in range(len(usuarioy)):
84             if i != "-" and i != " ":
85                 if usuariox[i] == usuarioy[j]:
86                     resultado.append(usuariox[i])
87     return resultado
88
89 #####
```




2.4. PrediccionSimple(simi[pearson | coseno | euclidea], vecinos, usuy).

Una vez que se recoge la métrica, lo siguiente al recoger los números de los vecinos es indicar el tipo de predicción a usar.

La predicción simple se basa en la fórmula:

Cálculo de Predicciones:

Calcular el valor desconocido $\hat{r}(u, i)$ (**predicción**) utilizando las puntuaciones asignadas a los ítems i de los usuarios v más parecidos (**vecinos más próximos**):

$$\hat{r}(u, i) = \frac{\sum_{v \in N_u^k} \text{sim}(u, v) \cdot r(v, i)}{\sum_{v \in N_u^k} |\text{sim}(u, v)|}$$

donde, N_u^k representa el conjunto de los k vecinos más próximos de u en términos de la función de similitud $\text{sim}(u, v)$.

En general necesitamos un número mínimo de k vecinos (no menos de 3).

Las variables que se usaron fueron:

- **numerador** y **denominador**, que corresponden al numerador y denominador donde se realizarán las operaciones correspondientes que satisfagan la fórmula.
- **usuy**, se recoge como valor ya que sólo se necesita de él.
- **prediccion[]**, es la lista que tendrá el resultado de la predicción que saldría.

El numerador realizará la multiplicación del valor de similitud que se haya pasado, ya sea pearson, coseno o euclídea, y la calificación del usuario **y** del objeto **i**.

Guardándose dicho resultado en la lista prediccion[].

Quedando el código tal que:



```
186 #####
187
188 def PrediccionSimple(similitud, vecinos, usuy):
189     prediccion = []
190     numerador = 0.0
191     denominador = 0.0
192     for i in range(len(usuy)):
193         if i < vecinos: #Nku conjunto de los k vecinos, a medias
194             numerador = similitud*i
195             denominador = abs(similitud)
196     prediccion = numerador/denominador
197     return prediccion
198
199
200 #####
201
```

2.4. PrediccionDMedia(simi[pearson | coseno | euclidea], vecinos, usux, usuy).

Por otro lado la predicción de distancia con la media, como su nombre indica, tendrá que ver con la media, y por ello la diferencia entre ella y la predicción simple, es que a ésta se le pasa un parámetro de más y es el usuario x.

Todo ello para basarse en la fórmula:

Cálculo de Predicciones:

La predicción con simples promedios no tiene en cuenta las desviaciones

El cálculo de predicciones considerando la **diferencia con la media** es como sigue:

$$\hat{r}(u, i) = \bar{r}(u) + \frac{\sum_{v \in N_u^k} sim(u, v) \cdot (r(v, i) - \bar{r}(v))}{\sum_{v \in N_u^k} |sim(u, v)|}$$

donde \bar{r} representa la media de puntuaciones.

- Solución para compensar las diferencias de interpretación: incluir la media del usuario activo y del vecindario.
- Lo mismo ocurre en la medida de similitud por correlación de Pearson respecto al coseno, más robusta a las desviaciones.

En una recomendación los ítems con mejores puntuaciones encontradas son recomendados.



Las variables que se usaron fueron:

- **numerador** y **denominador**, que corresponden al numerador y denominador donde se realizarán las operaciones correspondientes que satisfagan la fórmula.
- **usuy** y **usux**, los usuarios implicados.
- **mediax** y **mediay**, que tiene la media del usuario x e y.
- **prediccion[]**, es la lista que tendrá el resultado de la predicción que saldrá.

El numerador realizará la multiplicación del valor de similitud que se haya pasado, ya sea pearson, coseno o euclídea, y la calificación del usuario **y** del objeto **i** sobre su **media**.

Por su parte, el denominador contendrá el valor absoluto de la similitud.

Guardándose dicho resultado en la lista `prediccion[]`. La lista tendrá la operación que se le realiza al numerador / denominador + la media del usuario x.

Quedando el código tal que:

```
200 #####
201
202 def PrediccionDMedia(similitud, vecinos, usux, usuy):
203     prediccion = []
204     mediax, mediay = Media(usux, usuy)
205     numerador = 0.0
206     denominador = 0.0
207     for i in range(len(usuy)):
208         if i < vecinos: #Nku conjunto de los k vecinos, a medias
209             numerador = similitud*(i-mediay)
210             denominador = abs(similitud)
211         prediccion = numerador/denominador
212         prediccion = prediccion + mediax
213     return prediccion
214
215
216 #####
217
```

Finalizando así, todo el desarrollo y explicación del programa.

3. Funcionamiento.

El programa en mi caso, y como se explicó anteriormente, se ejecutaría:

```
> python sistema-recomenda.py matriz.txt
```

Mostrándose un ejemplo aleatorio, tal que:



1) Se ejecuta

```
andreacc@DESKTOP-EMSOIU9:/mnt/c/Users/andre/OneDrive/Escritorio$ python sistema-recomenda.py matriz.txt

OPCIONES DISPONIBLES
1. Correlacion de Pearson
2. Distancia coseno
3. Distancia Euclidea
4. Salir
Elige una opcion
Introduce un numero entero: █
```

2) Se elige la métrica Distancia de Coseno

```
Opcion 2: 'Distancia coseno'

+++ Matriz relacion usux/usuy de los usuarios +++
('1 1 1 1', '3 3 1 1')
('3 3 1 1', '2 - 3 -')
('2 - 3 -', '1 1 2 2')


('Matriz original de utilidad: ', [['1 1 1 1\r\n', '3 3 1 1\r\n', '2 - 3 -\r\n', '1 1 2 2']])
('usux: ', ['1 1 1 1\r\n', '3 3 1 1\r\n', '2 - 3 -\r\n'])
('usuy: ', ['3 3 1 1\r\n', '2 - 3 -\r\n', '1 1 2 2'])

('Distancia Coseno: ', 0.26666666666666666)

Correlacion directa
('Similitud: ', None)

INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS
Introduce un numero entero: █
```

3) Se Comprueba la matriz

```
C: > Users > andre > OneDrive > Escritorio >  matriz.txt

1    1 1 1 1
2    3 3 1 1
3    2 - 3 -
4    1 1 2 2
```

4) Se escoge 3 vecinos y Predicción Simple



```
Correlacion directa
('Similitud: ', None)

INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS
Introduce un numero entero: 3
INTRODUCE EL TIPO DE PREDICCION
1. Prediccion simple
2. Diferencia con la media
Introduce un numero entero: 1

Opcion 1: 'Prediccion simple'

('Prediccion: ', 2.0)
INTRODUCE EL TIPO DE PREDICCION
1. Prediccion simple
2. Diferencia con la media
Introduce un numero entero: █
```

4. Conflictos.

En este apartado se relatan algunos conflictos o problemáticas que me conllevó la realización de la práctica.

Por un lado, problemas con el paso por parámetros de la matriz y el poder trabajar con ella, lo cual se pudo solucionar.

Por otro lado, tuve problemas al sacar la media debido a que me daba errores de conversiones y operaciones entre distintos tipos de las variables, y las cuales me dificultaron a la hora de realizar dicho programa.

5. Referencias.

1. <https://stackoverflow.com/questions/17751322/python-2-attributeerror-list-object-has-no-attribute-strip>
2. <https://stackoverflow.com/questions/50735626/typeerror-list-object-is-not-an-iterator>
3. <https://es.stackoverflow.com/questions/69610/c%C3%B3mo-convertir-cada-l%C3%A9nea-de-un-archivo-de-texto-en-una-lista-utilizando-python>
4. <https://es.stackoverflow.com/questions/93202/problema-en-la-asignacion-de-matrices-en-python>
5. <https://www.it-swarm-es.com/es/python/como-vaciar-una-lista-en-python/967038412/>
6. <https://stackoverflow.com/questions/19480060/python-typeerror-unsupported-operand-types-for-str-and-float>