Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Gestión del Conocimiento enlas Organizaciones

Sistemas de Recomendación

Jefry Izquierdo Hernández
(alu0100996944@ull.edu.es)

Índice:

1. Introducción.

En esta práctica se quiere implementar un sistema de recomendación, el cuál maneja las métricas de: **Correlación de Pearson**, **Distancia Coseno** y **Distancia Euclídea**, a continución 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:
`python3 [program.py] [matriz.txt]`

El programa en python se llama: ***sistema-recomendacion.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.

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, yaque es más visual para poder trabajar con los datos.

Como paso previo se creó un método que pide un número entero que se comprueba, en caso contrario mostrará un mensaje de error.

```
###################

# Pedir numero, control de errores al recibir
def pedirNumeroEntero():
    correcto = False
    num = 0
    while(not correcto):
        try:
            num = int(input("Introduce un numero entero: "))
            correcto=True
        except ValueError:
            print('Error, introduce un numero entero')
    return num
salir = False
opcion = 0

###################
```

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, cuyo mínimo tiene que ser de 3. Y finalmente, se recoge el tipo de Predicción:

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

Una vez elegida la métrica, se procesa 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. Posteriormente, con la llamada a la función correspondiente a la métrica deseada: ***CoeficientePearson(usux, usuy), DCoseno(usux, usuy)*** o ***DEuclidea(usux, usuy)***. Sacando finalmente en dicho apartado el resultado de la métrica junto a un mensaje si la similitud cumple con lo establecido, para ello se llaman a las funciones: ***SimilitudPearson(resultadopearson), SimilitudCoseno(resultadocoseno)***.

Mostrándose así el código:

```
# Menu de opciones
while not salir:
    print("OPCIONES DISPONIBLES")
    print ("1. Correlacion de Pearson")
    print ("2. Distancia coseno")
    print ("3. Distancia Euclidea")
    print ("4. Salir")

    print ("Elige una opcion")
    opcion = pedirNumeroEntero()
    print("###################################")
    if opcion == 1:
        print ("\nOpcion 1: 'Correlacion de Pearson'\n")

        # Matriz usuarios/recomendacion
        matriz = sys.argv[1]
        usux, usuy = LeerFichero()
        resultadopearson = CoeficientePearson(usux, usuy)
        print("\n")
        print("Coeficiente de Pearson: ", resultadopearson)
        print("\n")
        print("Similitud: ", SimilitudPearson(resultadopearson))
        print("\n")

    elif opcion == 2:
        print ("\nOpcion 2: 'Distancia coseno'\n")

        # Matriz usuarios/recomendacion
        matriz = sys.argv[1]
        usux, usuy = LeerFichero()
        resultadocoseno = DCoseno(usux, usuy)
        print("\n")
        print("Distancia Coseno: ", resultadocoseno)
        print("\n")
        print("Similitud: ", SimilitudCoseno(resultadocoseno))
        print("\n")

    elif opcion == 3:
        print("\nOpcion 3: 'Distancia Euclidea'\n")

        # Matriz usuarios/recomendacion
        matriz = sys.argv[1]
        usux, usuy = LeerFichero()
        resultadoeuclidea = DEuclidea(usux, usuy)
        print("\n")
        print("Distancia Euclidea: ", resultadoeuclidea)
        print("\n")

    elif opcion == 4:
        salir = True
    else:
        print ("Introduce un numero entre 1 y 3")
        print("###################################")

    while True:
        #numvecinos = 0
```

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, 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.

```
    while True:
        print("INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS")
        vecinos = pedirNumeroEntero()
        if vecinos >= 3:
            break
        print("Introduce mas de 2 vecinos, minimo 3\n")


    while True:
        print("INTRODUCE EL TIPO DE PREDICCION")
        print ("1. Prediccion simple")
        print ("2. Diferencia con la media")
        prediccion = pedirNumeroEntero()
        if prediccion == 1:
            print ("\nOpcion 1: 'Prediccion simple'\n")
            if opcion == 1: #Correlacion de Pearson
                simipearson = CoeficientePearson(usux, usuy)
                prediccionspearson = PrediccionSimple(simipearson, vecinos, usuy)
                print("\n")
                print("Prediccion: ", prediccionspearson)
            elif opcion == 2: #Distancia Coseno
                simicoseno = DCoseno(usux, usuy)
                prediccionscoseno = PrediccionSimple(simicoseno, vecinos, usuy)
                print("\n")
                print("Prediccion: ", prediccionscoseno)
            elif opcion == 3: #Distancia Euclidea
                simieuclidea = DEuclidea(usux, usuy)
                prediccionseuclidea = PrediccionSimple(simieuclidea, vecinos, usuy)
                print("\n")
                print("Prediccion: ", prediccionseuclidea)
            else:
                print("Error con la prediccion simple")
        elif prediccion == 2:
            print ("\nOpcion 2: 'Diferencia con la media'\n")
            if opcion == 1: #Correlacion de Pearson
                simipearson = CoeficientePearson(usux, usuy)
                predicciondpearson = PrediccionDMedia(simipearson, vecinos, usux, usuy)
                print("\n")
                print("Prediccion: ", predicciondpearson)
            elif opcion == 2: #Distancia Coseno
                simicoseno = DCoseno(usux, usuy)
                predicciondcoseno = PrediccionDMedia(simicoseno, vecinos, usux, usuy)
                print("\n")
                print("Prediccion: ", predicciondcoseno)
            elif opcion == 3: #Distancia Euclidea
                simieuclidea = DEuclidea(usux, usuy)
                predicciondeuclidea = PrediccionDMedia(simieuclidea, vecinos, usux, usuy)
                print("\n")
                print("Prediccion: ", predicciondeuclidea)
            else:
                print("Error con la prediccion diferencia con la media")
        else:
            print("###################################")
            print ("Introduce un numero, 1 o 2")
            print("###################################")
print ("Fin")
```

En este apartado entraremos en detalle en las funciones nombradas previamente.

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.

```
# Tratamiento de argumentos
# Leer linea a linea, guardando valoracion usuario
def LeerFichero():
    print ("+++ Matriz relacion usux/usuy de los usuarios ++++")
    datosusuario = []
    usux = []
    usuy = []
    with open("matriz.txt") as fname:
        usuarios = fname.readlines()
    for i in range(len(usuarios)):
        if i+1 < len(usuarios):
            usux.append(usuarios[i])
            usuy.append(usuarios[i+1])
            # Se llama al metodo para limpiar blancos y espacios
            newusux, newusuy = LimpiarEspacios(usux, usuy)
            print(newusux, newusuy)
            CoeficientePearson(newusux, newusuy)

    datosusuario.append(usuarios)
    print("\n")
    print("Matriz original de utilidad: ", datosusuario)
    print("usux: ", usux)
    print("usuy: ", usuy)
    return usux, usuy

print("\n")
```

Debido a que habían elementos que hacían que el código fuera engorros se implementó 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 esta función tal que:

```
####################

# Limpiar espacios en blanco
def LimpiarEspacios(usux, usuy):
    for i in range(len(usux)):
        if i < len(usux):
            newusux = usux[i].strip()
            newusuy = usuy[i].strip()
    return newusux, newusuy
```

Lo siguiente es implementar la función correspondiente para Pearson, Coseno y Euclídea.

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
- $\overline{r}(u)$ 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 $S_{uv} = S_u \cap S_v$

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

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.

```
####################

def CoeficientePearson(usuariox, usuarioy):
    pearson = []
    # Calificaciones del usuario sobre objeto i
    vcalificacionx = CalificacionObjeto(usuariox)
    vcalificaciony = CalificacionObjeto(usuarioy)

    # Media de calificaciones de los usuarios
    mediax, mediay = Media(vcalificacionx,vcalificaciony)


    numerador = 0.0
    xpow = 0.0
    ypow = 0.0
    for i in range(len(usuariox)):
        numerador += i-mediax*i-mediay
    for i in range(len(usuariox)):
        xpow += math.pow(i-mediax,2)
    for i in range(len(usuariox)):
        ypow += math.pow(i-mediay,2)
    # Producto de las raices cuadradas de los usuarios x e y
    denominador = math.sqrt(xpow*ypow)
    pearson = numerador/denominador
    return pearson


####################
```

La función **CalificacionObjeto()** sería:

```
####################

# Calificaciones del usuario sobre el objeto
def CalificacionObjeto(usuario):
    resultado = []
    for i in range(len(usuario)):
        if i != "-" and i != " ":
            resultado.append(usuario[i])
    return resultado


####################
```

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

Y la función **Media()** sería:

```
####################

# Media
def Media(usux,usuy):
    sumax = 0
    sumay = 0
    lenvector = 0
    for i in range(len(usux)):
        for j in range(len(usuy)):
            if i < len(usux) and j < len(usuy):
                aux=int(i)
                aux2=int(j)
                sumax += aux
                sumay += aux2
                lenvector = lenvector+1
            if lenvector==0  or i =="-" or j =="-" or i ==" " or j ==" ":
                return 0,0
            else:
                xresult = float(sumax)/lenvector
                yresult = float(sumay)/lenvector
                i+1,j+1
    return xresult, yresult


####################
```

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.

```
###################

def SimilitudPearson(valor):
    if valor == 1:
        print("Correlacion directa perfecta")
    elif 0 < valor < 1:
        print("Correlacion directa")
    elif valor == 0:
        print("No hay correlacion")
    elif -0 < valor < 0:
        print("Correlacion inversa")
    elif valor == -1:
        print("Correlacion inversa perfecta")
    else:
        print("No se encuentra similitud")


###################
```

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

2.2. DCoseno(usux, usuy).

Apoyada en la fórmula matemática:

Medidas de similitud: Distancia Coseno

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

```
def DCoseno(usuariox, usuarioy):
    coseno = []
    # Calificaciones del usuario sobre objeto i
    vcalificacionx = CalificacionObjeto(usuariox)
    vcalificaciony = CalificacionObjeto(usuarioy)

    numerador = 0.0
    xpow = 0.0
    ypow = 0.0
    for i in range(len(vcalificacionx)):
        for j in range(len(vcalificaciony)):
            numerador = i*j
            xpow += math.pow(i,2)
            ypow += math.pow(j,2)
            denominador = math.sqrt(xpow*ypow)
    coseno = numerador/denominador
    return coseno
```

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

```
####################

def SimilitudCoseno(valor):
    if valor == 1:
        print("Correlacion directa perfecta")
    elif 0 < valor < 1:
        print("Correlacion directa")
    elif valor == 0:
        print("No hay correlacion")
    else:
        print("No se encuentra similitud")
```

Y la cual se basó en la fórmula:

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.

2.3. DEuclidea(usux, usuy).

Basada 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**.
- **euclidea[ ]**, 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:

```
def DEuclidea(usuariox, usuarioy):
    deuclidea = []
    # Calificaciones del usuario sobre objeto i
    vcalificacionx = CalificacionObjeto(usuariox)
    vcalificaciony = CalificacionObjeto(usuarioy)

    # Misma calificaciones de ambos usuarios sobre un objeto i
    vcalificacionxy = CalificacionIgualObjeto(vcalificacionx, vcalificaciony)

    xypow = 0.0
    for i in range(len(vcalificacionx)):
        for j in range(len(vcalificaciony)):
            xypow = math.pow(i-j,2)
    deuclidea = math.sqrt(xypow)
    return deuclidea
```

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:

```
####################

# Misma calificaciones de los dos usuarios sobre un objeto
def CalificacionIgualObjeto(usuariox, usuarioy):
    resultado = []
    for i in range(len(usuariox)):
        for j in range(len(usuarioy)):
            if i != "-" and i != " ":
                if usuariox[i] == usuarioy[j]:
                    resultado.append(usuariox[i])
    return resultado

####################
```

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 usuaros v más parecidos (**vecinos más próximos**):

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

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

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:

```python
def PrediccionSimple(similitud, vecinos, usuy):
    prediccion = []
    numerador = 0.0
    denominador = 0.0
    for i in range(len(usuy)):
        if i < vecinos: #Nku conjunto de los k vecinos, a medias
            numerador = similitud*i
        denominador = abs(similitud)
    prediccion = numerador/denominador
    return prediccion
```

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:

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

donde $\overline{r}$ 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 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** 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:

```
def PrediccionDMedia(similitud, vecinos, usux, usuy):
    prediccion = []
    mediax, mediay = Media(usux,usuy)
    numerador = 0.0
    denominador = 0.0
    for i in range(len(usuy)):
        if i < vecinos: #Nku conjunto de los k vecinos, a medias
            numerador = similitud*(i-mediay)
        denominador = abs(similitud)
    prediccion = numerador/denominador
    prediccion = prediccion + mediax
    return prediccion
```

Dando por finalizado todo el desarrollo y toda la explicación del código de este programa.

3. Funcionamiento y ejemplos.

El programa en mi caso, y como se explicó anteriormente, se ejecutaría:
`# python3 sistema-recomendacion.py matriz-[num]-[num]-[num].txt`

EJEMPLO 1: Matriz : **utility-matrix-5-10-2.txt** (https://github.com/cexposit/ull-gco/blob/main/examples-utility-matrices/utility-matrix-5-10-2.txt)

- **Métrica**: Correlación de Pearson
- **Vecinos**: 3
- **Predicción**: Simple

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL
jefry@jefry-VirtualBox:~/Documentos/GCO/Practicas$ python3 sistema-recomendacion.py utility-matrix-5-10-2.txt


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

Opcion 1: 'Correlacion de Pearson'

+++ Matriz relacion usux/usuy de los usuarios ++++
5 3 4 4 - 3 1 2 3 3
3 1 2 3 3 4 3 4 3 5
4 3 4 3 5 3 3 1 5 4
3 3 1 5 4 1 5 5 2 1


Matriz original de utilidad:  [['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']]
usux:  ['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n']
usuy:  ['3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']


Coeficiente de Pearson:  -1.8


No se encuentra similitud
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:
```

EJEMPLO 2: Matriz : **utility-matrix-10-25-3.txt** (https://github.com/cexposit/ull-gco/blob/main/examples-utility-matrices/utility-matrix-10-25-3.txt)

- **Métrica**: Distancia coseno
- **Vecinos**: 5
- **Predicción**: Simple

```
jefry@jefry-VirtualBox:~/Documentos/GCO/Practicas$ python3 sistema-recomendacion.py utility-matrix-10-25-3.txt


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

Opcion 2: 'Distancia coseno'

+++ Matriz relacion usux/usuy de los usuarios ++++
5 3 4 4 - 3 1 2 3 3
3 1 2 3 3 4 3 4 3 5
4 3 4 3 5 3 3 1 5 4
3 3 1 5 4 1 5 5 2 1


Matriz original de utilidad:  [['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']]
usux:  ['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n']
usuy:  ['3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']


Distancia Coseno:  0.16071428571428573


Correlacion directa
Similitud:  None


INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS
Introduce un numero entero: 5
INTRODUCE EL TIPO DE PREDICCION
1. Prediccion simple
2. Diferencia con la media
Introduce un numero entero: 1

Opcion 1: 'Prediccion simple'



Prediccion:  3.0
INTRODUCE EL TIPO DE PREDICCION
1. Prediccion simple
2. Diferencia con la media
Introduce un numero entero:
```

EJEMPLO 3: Matriz : **utility-matrix-5-10-9.txt** (https://github.com/cexposit/ull-gco/blob/main/examples-utility-matrices/utility-matrix-5-10-9.txt)

- **Métrica**: Distancia Euclídea
- **Vecinos**: 4
- **Predicción**: Distancia con la media

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

Opcion 3: 'Distancia Euclidea'

+++ Matriz relacion usux/usuy de los usuarios ++++
5 3 4 4 - 3 1 2 3 3
3 1 2 3 3 4 3 4 3 5
4 3 4 3 5 3 3 1 5 4
3 3 1 5 4 1 5 5 2 1


Matriz original de utilidad:  [['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']]
usux:  ['5 3 4 4 -\n', '3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n']
usuy:  ['3 1 2 3 3\n', '4 3 4 3 5\n', '3 3 1 5 4\n', '1 5 5 2 1']


Distancia Euclidea:  0.0


INTRODUCE EL NUMERO DE VECINOS CONSIDERADOS
Introduce un numero entero: 4
INTRODUCE EL TIPO DE PREDICCION
1. Prediccion simple
2. Diferencia con la media
Introduce un numero entero: 2

Opcion 2: 'Diferencia con la media'

Traceback (most recent call last):
  File "sistema-recomendacion.py", line 352, in <module>
    prediccion_deuclidea = PrediccionDMedia(simieuclidea, vecinos, usux, usuy)
  File "sistema-recomendacion.py", line 200, in PrediccionDMedia
    prediccion = numerador/denominador
ZeroDivisionError: float division by zero
jefry@jefry-VirtualBox:~/Documentos/GCO/Practicas$
```

4. Conflictos.

Por un lado, me ha dado problemas 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, lo cual me dificultó mucho a la hora de hacer el 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%ADnea-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