Sistemas de Recomendación

MÁSTER UNIVERSITARIO EN CIBERSEGURIDAD
E INTELIGENCIA DE DATOS

Alba Cruz Torres | Escuela de Doctorado y Estudios de Posgrado | 15/09/2022

INDICE

1. INTRODUCCIÓN

Para la realización de esta práctica se va a implementar un **Sistema de Recomendación basado en Contenido**, el cual es un sistema que utiliza las características del artículo o producto para hacer las recomendaciones, a diferencia de otros sistemas que se centran en el usuario y su comportamiento.

En esta práctica se pretende tener un fichero de texto como entrada, en el que se almacena un conjunto de documentos de texto que se desean recomendar al lector. Para este cometido, se ha tenido en cuenta los gustos del lector, de manera que se puedan recomendar textos con temas similares a los que le ha gustado al lector.

Para poder hacer dichas recomendaciones, se necesita realizar el cálculo de la similitud entre cada par de documentos, teniendo en cuenta que dicho cálculo podrá tomar valores entre 0-1 y que cuanto más cercano a 1 sea ese valor, más similares serán los documentos entre sí.

Por tanto, el objetivo de esta práctica es el de realizar ese cálculo de la similitud de los documentos y entender cómo se realiza dicho cálculo. Para poder llevarlo a cabo, se hace uso de la medida de similitud del coseno y esto es posible haciendo uso del formato de codificación ***TF-IDF (Frecuencia del Término – Frecuencia Inversa del Documento)***. Es decir, los documentos de texto son codificados ***TF-IDF*** como vectores en un espacio Euclidiano multidimensional.

$$TF(i,j) = \frac{freq(i,j)}{maxOthers(i,j)} \qquad IDF(i) = log\frac{N}{n(i)}$$

$$TF\text{-}IDF(i,j) = TF(i,j) * IDF(i)$$

En el modelo ***TF-IDF*** el documento se representa como un vector de los pesos ***TF-IDF***. Cada una de las partes con la que se calculan dichos pesos son las siguientes:

- **Frecuencia del Término (*TF*):** Hace referencia a cuántas veces se repite una palabra o término en el texto.
- **Frecuencia Inversa del Documento (*IDF*):** Se encarga de reducir el peso de las palabras clave que aparecen muy a menudo en todos los documentos.

Para la implementación de esta práctica se ha decidido utilizar el lenguaje de programación *Python* en la versión 3.8.10 y se ha almacenado el código fuente en el siguiente enlace de GitHub:

https://github.com/alu0100889635/TAAD/tree/main/P2SR.

2. IMPLEMENTACIÓN

2.1. INSTALACIÓN DE LIBRERÍAS

Una vez entendido el problema que se quiere resolver, se procede a la implementación de la solución. Por consiguiente, lo primero que se debe hacer es instalar las librerías de *Python* necesarias para el correcto funcionamiento del programa. Para ello, se ejecuta el siguiente comando:

```
alba@DESKTOP-10234JR:~/TAAD/P2SR$ python3 instalacion.py
```

En el fichero *instalation.py* se encuentran todas esas librerías que se deben instalar y los comandos requeridos para ese cometido. En este caso, el número de librerías necesarias es pequeño y, por tanto, no debería tardar mucho tiempo en ejecutarse. Dichas librerías son:

```
import sys
import subprocess

# implement pip as a subprocess:
subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'pandas'])

subprocess.check_call([sys.executable, '-m', 'pip', 'install', "-U",
'scikit-learn'])

subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'numpy'])
```

- ***Pandas:*** Librería de *Python* especializada en el manejo y análisis de estructuras de datos.
- ***Scikit-learn:*** Librería de *Python* para aprendizaje automático.
- ***Numpy:*** Librería de *Python* para crear vectores y matrices grandes multidimensionales, además de proveer de un conjunto de funciones matemáticas de alto nivel para operar con ellas.

2.2. LECTURA DE FICHERO DE DOCUMENTOS

Después de terminar el proceso de instalación, se procede a la lectura del archivo de texto donde se almacena el conjunto de documentos. El archivo de texto en el que se almacenan los documentos tiene el formato que se muestra a continuación:

1: Aromas include tropical fruit, broom, brimstone, and dried herb. The palate isn't overly expressive, offering unripened apple, citrus and dried sage alongside brisk acidity. /1

2: This is ripe and fruity, a wine that is smooth while still structured. Firm tannins are filled out with juicy red berry fruits and freshened with acidity. It's already drinkable, although it will certainly be better from 2016. /1

3: Tart and snappy, the flavors of lime flesh and rind dominate. Some green pineapple pokes through, with crisp acidity underscoring the flavors. The wine was all stainless-steel fermented. /1

4: Pineapple rind, lemon pith and orange blossom start off the aromas. The palate is a bit more opulent, with notes of honey-drizzled guava and mango giving way to a slightly astringent, semidry finish. /1

5: Much like the regular bottling from 2012, this comes across as rather rough and tannic, with rustic, earthy, herbal characteristics. Nonetheless, if you think of it as a pleasantly unfussy country wine, it's a good companion to a hearty winter stew. /0

6: Blackberry and raspberry aromas show a typical Navarran whiff of green herbs and, in this case, horseradish. In the mouth, this is full bodied, with tomatoes acidity. Spicy, herbal flavors complement dark plum fruit, while the finish is fresh but grabby. /0

7: Here's a bright, informal red that opens with aromas of candied berry, white pepper and savory herb that carry over to the palate. It's balanced with fresh acidity and soft tannins. /0

10: Tart and happy, the flavors of crime flesh and rind dominate. Some green pineapple pokes through, with crisp acidity underscoring the flavors. The wine was all stainless-steel fermented. /0

20: Pineapple kind, lemon pitch and banana blossom start off the aromas. The palate is a bit more opulent, with notes of honey-drizzled guava and mango giving way to a slightly astringent, semidry finish. /0

Cada uno de los textos se encuentra en una línea diferente y están numerados. Además, en cada documento se proporciona su estado, es decir, si le ha gustado (1) o no (0) o si todavía no lo ha leído (N). Por tanto, el formato genérico del archivo de texto sería el que sigue:

NºDoc: Documento /Estado

Cuando se ejecute el software, el nombre del archivo de texto se pasará a través de la línea de comandos:

```
alba@DESKTOP-10234JR:~/TAAD/P2SR$ python3 recommenderSys.py texts.txt
```

Dicho archivo de texto será abierto y leído por el software de la siguiente manera:

```
with open(sys.argv[1], 'r') as f:
    liked = [line.rstrip() for line in f]
```

Con esas líneas de código lo que se hace es coger el nombre que se introduce por la línea de comandos y se lee el fichero que se corresponde con ese nombre. Para que esto funcione, el archivo de texto debe estar en la misma ruta que el fichero de código. Una vez leído, guarda en una posición de la lista *liked* cada una de las líneas del archivo, de manera que se tiene almacenado cada documento en una posición distinta de la lista. El siguiente paso será el de recorrer la lista y separar cada documento de su número y de su estado (0, 1, N) y almacenarlos en un *DataFrame*.

Esta división se hace gracias a una expresión regular, en la que se almacena, por un lado, todo lo que hay antes del primer ":" que encuentre y, por otro lado, todo lo que hay después del separador "/", de forma que se tenga el número del documento, el documento y su estado por separado:

```
documents = list(map(lambda x: re.split(r"\:\s|\/", x, 2), liked))
dfdocs = pd.DataFrame(documents, columns=['DocNumb', 'Document', 'Likes'])
print(dfdocs)
```

Después de haber creado el *DataFrame* con los datos obtenidos del archivo de texto, el resultado queda de la siguiente manera:

```
alba@DESKTOP-10234JR:~/TAAD/P2SR$ python3 recommenderSys.py texts.txt
   DocNumb                                           Document Like
0        1  Aromas include tropical fruit, broom, brimston...    1
1        2  This is ripe and fruity, a wine that is smooth...    1
2        3  Tart and snappy, the flavors of lime flesh and...    1
3        4  Pineapple rind, lemon pith and orange blossom ...    1
4        5  Much like the regular bottling from 2012, this...    0
5        6  Blackberry and raspberry aromas show a typical...    N
6        7  Here's a bright, informal red that opens with ...    N
7       10  Tart and happy, the flavors of crime flesh and...    N
8       20  Pineapple kind, lemon pitch and banana blossom...    N
```

En cada posición del *DataFrame* se ha almacenado el número del documento junto con el contenido del documento y también su estado.

2.3. CÁLCULO DE *TF-IDF*

Una vez que se tienen todos los documentos, se puede comenzar a calcular el ***TF-IDF*** de cada uno de ellos. Para este fin, se ha creado una función a la que se le pasa un documento por parámetro. En esta función, se hace uso de un método de la librería *Scikit Learn*, llamado ***TfidfVectorizer,*** que se encarga de convertir una colección de documentos en una matriz de pesos ***TF-IDF***. Es decir, este método se encarga de calcular tanto la Frecuencia del Término y la Frecuencia Inversa del Documento como la codificación *TF-IDF* de los documentos. Para que el análisis de los documentos sea más sencillo y se centre en las palabras que aportan información para ser comparada, se prescinde de las *stop_words*. En otras palabras, se prescinde de las preposiciones, los conectores y demás palabras que no aportan información relevante al tema del documento. Dicha función, devuelve la matriz de los pesos *TF-IDF* y las palabras clave de cada documento.

```
def vectorizeDocs(doc):
    vectorizer = TfidfVectorizer(stop_words = "english")
    return vectorizer.fit_transform(doc), vectorizer.get_feature_names_out()

tfidfmatrix, words = vectorizeDocs(dfdocs['Document'])
```

Luego, se realiza el desglose de los pesos de cada uno de los términos que componen cada documento. Para ello, basta con recorrer el vector *tfidfmatrix* devuelto por la función y guardarlo en una lista. Sin embargo, como se comentó anteriormente, se ignoran las *stop_words* y, por tanto, sus pesos son iguales a 0.0. Es por esto por lo que, se hace un recorrido del conjunto de pesos y se guardan solo aquellos que son distintos a 0.0. Es decir, aquellos pesos correspondientes a las palabras clave.

```
tfidfdoc = []
for i in tfidfmatrix.toarray():
    tfidfdocnonzero = []
    for x in range(0, len(i)):
        if(i[x]) != 0.0:
            tfidfdocnonzero.append(i[x])
    tfidfdoc.append(tfidfdocnonzero)
```

A continuación, se calculan los términos de cada documento y sus posiciones o índices dentro del documento.

```
words = []
positions = []
for i in test:
    X, word = vectorizeDocs(i)
    position = []
    for w in word:
        position.append(i[0].lower().find(w))
    positions.append(position)
    words.append(word)
```

Finalmente, después de haber calculado los términos de cada documento, sus posiciones y sus codificaciones *TF-IDF*, se procede a añadir estos datos al *DataFrame* que se creó con anterioridad, de manera que su representación es la que sigue:

	DocNumb	Term Ind
0	1	[165, 149, 120, 0, 38, 159, 31, 127, 53, 89, 2...
1	2	[221, 146, 114, 209, 196, 168, 88, 71, 131, 12...
2	3	[109, 103, 52, 176, 21, 37, 67, 32, 73, 83, 47...
3	4	[60, 172, 84, 38, 122, 192, 147, 131, 116, 16,...
4	5	[36, 22, 116, 47, 215, 189, 101, 210, 230, 109...
5	6	[158, 25, 0, 136, 90, 189, 200, 227, 181, 237,...
6	7	[156, 46, 136, 64, 9, 56, 105, 150, 95, 17, 35...
7	10	[109, 31, 103, 52, 176, 20, 37, 67, 9, 73, 83,...
8	20	[61, 173, 32, 85, 39, 123, 193, 148, 132, 117,...

	Terms	TF-IDF	Like
[acidity, alongside, apple, aromas, brimstone,...	[0.11496011351981335, 0.2211150725356627, 0.22...	1
[2016, acidity, berry, better, certainly, drin...	[0.2513273056551915, 0.1306677797104477, 0.212...	1
[acidity, crisp, dominate, fermented, flavors,...	[0.145498592656102, 0.2363682934622162 7, 0.236...	1
[aromas, astringent, bit, blossom, drizzled, f...	[0.1497106102388572, 0.21839589446663235, 0.21...	1
[2012, bottling, characteristics, comes, compa...	[0.21752298346423554, 0.21752298346423554, 0.2...	0
[acidity, aromas, blackberry, bodied, case, co...	[0.11521315121180473, 0.12830411722668417, 0.2...	N
[acidity, aromas, balanced, berry, bright, can...	[0.13609752668233965, 0.15156145660499512, 0.2...	N
[acidity, crime, crisp, dominate, fermented, f...	[0.145498592656102, 0.27985299321617324, 0.236...	N
[aromas, astringent, banana, bit, blossom, dri...	[0.14745679222778163, 0.215108054014252, 0.254...	N

3. CÁLCULO DE LA SIMILITUD DEL COSENO

El siguiente paso es el de calcular la similitud del coseno entre cada par de documentos. Se mide la similitud entre dos vectores n-dimensionales basada en el ángulo que hay entre ellos. La fórmula de la similitud del coseno es la que se muestra a continuación:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$

Como se nombró en la introducción, el valor de la similitud del coseno puede oscilar entre 0-1, siendo más similares entre sí cuanto más cercano a 1 sea el valor. Por tanto, para hacer este cálculo, se ha hecho uso de la función llamada *cosine_similarity* que también ofrece la librería *Scikit Learn*. Dicha función realiza el cálculo de la similitud entre un conjunto de documentos, dando como resultado una matriz en la que cada fila y cada columna se corresponde con un documento.

```
cosine_similarities = cosine_similarity(tfidfmatrix, tfidfmatrix)
lowerTriangleMatrix = np.tril(cosine_similarities)

dfmatrix = pd.DataFrame(lowerTriangleMatrix)
```

Como se está haciendo una comparación del conjunto de todos los documentos consigo mismo, la diagonal principal resultará ser igual a 1, puesto que se está haciendo una comparación de cada documento consigo mismo. Además, al ser el mismo conjunto de documentos el que se compara, basta con que se use la matriz triangular superior o la inferior para sacar las similitudes del coseno, pues si se cogen ambas, se estarían repitiendo los resultados. En este caso, se ha decidido utilizar la matriz triangular inferior.

4. REPRESENTACIÓN DE RESULTADOS

Por último, se deben mostrar por pantalla los resultados del cálculo de la similitud entre cada par de documentos. Para ello, se debe tener en cuenta los documentos que le han gustado al lector y los que no ha leído aún. En primer lugar, se recorre la matriz triangular inferior donde se tienen los valores de similitud. Luego, se tendrá en cuenta si el estado es igual a "1", es decir, que le ha gustado y entonces se procede a ordenar los valores de la similitud del coseno de mayor a menor, con el fin de mostrar por pantalla de los documentos más similares a los menos similares al que le han gustado.

```
55  for i in range(0, len(dfmatrix.columns)):
56      if dfdocs.iloc[i]['Like'] == "1":
57          print("\nLe ha gustado el documento ", dfdocs.iloc[i]['DocNumb'],": ")
58          sorted_values = dfmatrix[i].sort_values(ascending = False)
59          for index, value in sorted_values.items():
60              if(round(value, 6) != 1.0):
61                  if(round(value, 6) >= 0.0):
62                      if(dfdocs.iloc[index]['Like'] == "N"):
63                          print("Documento ", dfdocs.iloc[index]['DocNumb'],
64                          " -> Similitud con documento", dfdocs.iloc[i]['DocNumb'], "= ", round(value, 6))
```

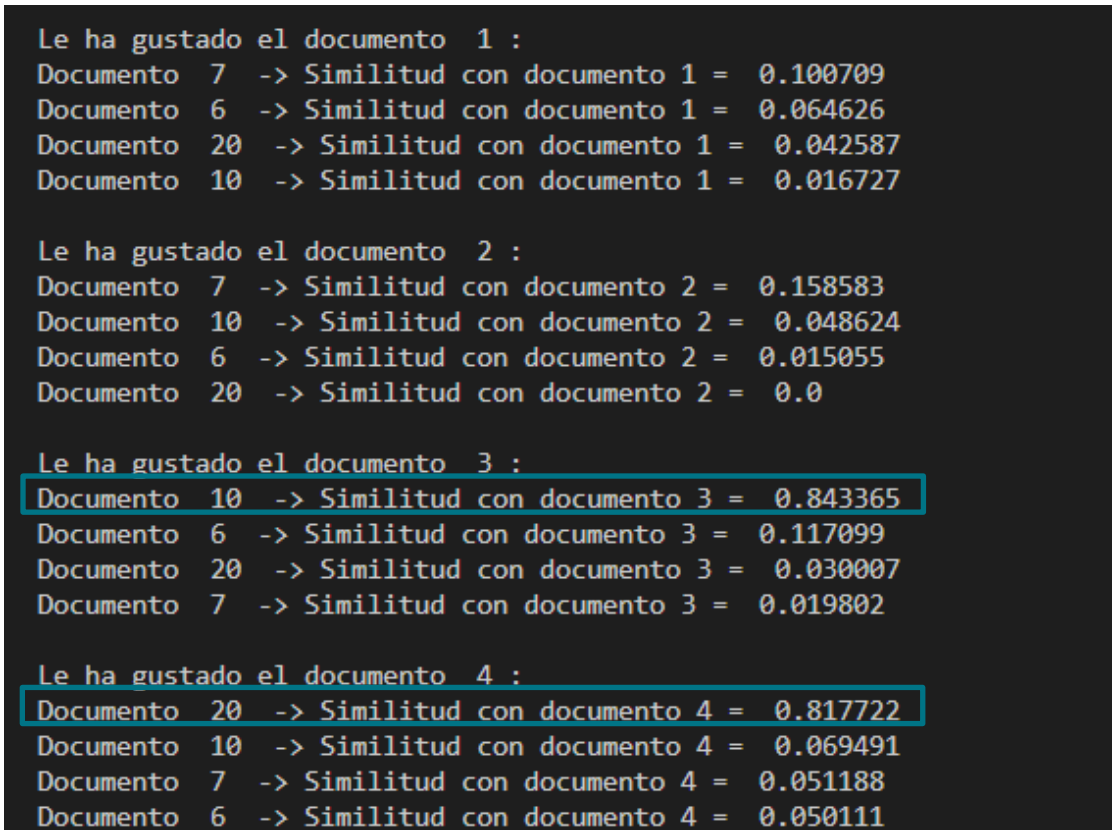
Como se puede ver en el bucle anterior, se descartan los valores igual a 1.0, puesto que son los valores que hacen referencia a la similitud de un documento consigo mismo. Ya que los valores presentaban bastantes decimales, se ha decidido redondearlos todos a 6 decimales, para que haya igualdad entre los números a la hora de compararlos.

Por otro lado, en la línea 61, se hace una comparación de si los valores son mayor o igual a un número concreto. Esta condición se añade en el código con el objetivo de mostrar los documentos más similares a los que le han gustado, es decir, los que tengan un valor de la similitud del coseno más cercanos a 1.0 (p. ej.: *round (value, 6) >= 0.8)*. Sin embargo, como en este caso lo que se desea es ver todos los resultados del cálculo de la similitud del coseno, se ha puesto que sea mayor o igual que 0.0, de manera que se muestren todos los valores.

Finalmente, el resultado de ejecutar el software con el conjunto de documentos sería el siguiente: se muestran los documentos que le han gustado al lector y luego se muestran sus similitudes con los otros documentos que aún no ha leído.

Como se puede observar en la siguiente imagen, se puede comprobar que se muestran los documentos a los que el lector a indicado que le gustan, que en este caso son el D1, D2, D3 y D4 y se muestran sus similitudes con los documentos no leídos aún por el lector, que son los documentos D6, D7, D10 y D20.

Si se pone el foco (recuadro azul) en los que tienen un valor de la similitud del coseno mayor o igual que 0.8, se acabaría recomendando al lector el D10 porque le ha gustado el D3 y el D20 porque le ha gustado el D4.



6. MATERIAL CONSULTADO

El material que se ha consultado para realizar este proyecto ha sido el siguiente:

- Apuntes de la asignatura en el Campus Virtual.
- Scikit – Learn: Machine Learning in Python
- Pandas Documentation Python