

## Módulo 8

# Sistema de recomendación de libros

Marta Bellón Castro  
Curso 2022-2023

Luis de la Ossa  
Máster en Ciencia de Datos e Ingeniería de Datos en la Nube  
Universidad de Castilla-La Mancha

## Índice

- [1. Introducción](#)
  - [Lectura de las revisiones](#)
  - [Etiquetas](#)
  - [Preparación de los datos](#)
- [2. Limpieza del texto](#)
- [3. Búsqueda por similitud de la representación tf-idf](#)
- [4. Recomendación basada en contenido](#)
  - [Selección de libros preferidos por el usuario](#)
  - [Búsqueda de libros similares](#)
  - [Priorización de resultados y recomendación](#)
- [5. Sistema híbrido de recomendación](#)
- [6. LDA y búsqueda por similitud](#)

```
In [1]: from IPython.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))

import warnings
warnings.filterwarnings('ignore')
```

## 1. Introducción

Los sistemas de recomendación basados en filtrado colaborativo utilizan de manera exclusiva los perfiles de votación de usuarios/items, y no consideran el contenido. Como se vió en clase, permiten obtener resultados que, si bien cualitativamente son aceptables, pueden llegar a desconcertar al no corresponderse con lo esperado.

En este proyecto se diseñará un pequeño sistema de recomendación de libros basado en contenido. Para ello, será necesario hacer uso de algunos de los conceptos relacionados con aprendizaje automático sobre información textual (*Text Mining*) que han sido tratados también a lo largo del módulo.

Como punto de partida, se partirá del conjunto de datos [goodbooks-10k](https://www.kaggle.com/zygmunt/goodbooks-10k) (<https://www.kaggle.com/zygmunt/goodbooks-10k>) disponible en [kaggle](https://www.kaggle.com) (<https://www.kaggle.com>). Éste contiene información relativa a 10000 libros obtenida de la red social [goodreads](http://goodreads.com) (<http://goodreads.com>), que actualmente es el sitio de referencia en la red para aficionados a la lectura. Además de títulos y autores, el conjunto de datos incluye votos y etiquetas aportadas por más de 53000 usuarios.

❗ Se han hecho algunas modificaciones con respecto a la base original para que sea menos tedioso manejar los distintos índices e identificadores.

```
In [2]: import pandas as pd
import numpy as np

df_goodreads = pd.read_csv('data/books.csv', sep="\t", index_col=0)
df_goodreads.head(2)
```

Out[2]:

	gr_book_id	gr_best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	title	...	ratings_count	work_ratings_count	work_book_id
--	------------	-----------------	---------	-------------	------	--------	---------	---------------------------	----------------	-------	-----	---------------	--------------------	--------------

0	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	The Hunger Games (The Hunger Games, #1)	...	4780653	4942365	
1	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	Harry Potter and the Sorcerer's Stone (Harry P...	...	4602479	4800065	

2 rows × 22 columns



## Lectura de las revisiones

Los datos de la tabla apenas contienen información relativa al contenido de cada libro. Sin embargo, es posible acceder a los resúmenes almacenados en la propia web [GoodReads](http://www.goodreads.com) ([www.goodreads.com](http://www.goodreads.com)) que, en cierto grado, aportan esta información. En el [anexo](#) se adjunta la función utilizada para obtenerlos.

El resumen de cada libro ha sido almacenado en un archivo de texto denominado `./data/overviews/bookid.txt`, donde *bookid* corresponde al identificador del libro (columna `book_id` de `df_goodreads`).

✍ Implementar una función, denominada `get_overview(gr_book_id)`, que reciba el identificador de un libro, lea el archivo de texto que contiene el resumen correspondiente y lo devuelva en un *String* (o devuelva `None` si este resumen no existe).

```
def get_overview(gr_book_id): try: ## COMPLETAR # except: return None
```

```
get_overview(320) # Cien años de soledad (Gabriel García Márquez)
```

```
In [3]: def get_overview(gr_book_id):
        try:
            with open(f"./data/overviews/{gr_book_id}.txt", "r", encoding="utf-8") as f:
                overview = f.read()
            return overview
        except FileNotFoundError:
            return None
```

```
get_overview(320) # Cien años de soledad (Gabriel García Márquez)
```

```
Out[3]: '(Book Jacket Status: Jacketed)The brilliant, bestselling, landmark novel that tells the story of the Buendia family, and chronicles the irreconcilable conflict between the desire for solitude and the need for love—in rich, imaginative prose that has come to define an entire genre known as "magical realism."'
```



✍ Crear una columna en `df_goodreads`, denominada `overview`, que contenga la revisión del libro correspondiente. Rellenar los valores vacíos de esa columna con un *String* de longitud cero (`""`).

```
In [4]: df_goodreads['overview'] = df_goodreads['gr_book_id'].apply(get_overview).fillna('')
df_goodreads.head(5)
```

Out[4]:

	gr_book_id	gr_best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	title	...	work_ratings_count	work_text_reviews_
book_id													
0	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	The Hunger Games (The Hunger Games, #1)	...	4942365	1
1	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	Harry Potter and the Sorcerer's Stone (Harry P...	...	4800065	
2	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0	Twilight	Twilight (Twilight, #1)	...	3916824	
3	2657	2657	3275794	487	61120081	9.780061e+12	Harper Lee	1960.0	To Kill a Mockingbird	To Kill a Mockingbird	...	3340896	
4	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0	The Great Gatsby	The Great Gatsby	...	2773745	

5 rows × 23 columns

☒

Etiquetas

En el conjunto de datos se proporcionan etiquetas relativas a cada libro que han sido aportadas por los usuarios. Esta información está incluida en dos archivos. El primero de ellos, `tags.csv` , contiene el identificador de cada etiqueta y el código correspondiente. El segundo, `book_tags.csv` contiene las etiquetas relativas a cada libro. Se almacenarán, respectivamente, en los *DataFrame* `df_tags` y `df_book_tags` .



```
In [5]: df_tags = pd.read_csv('./data/tags.csv')
df_book_tags = pd.read_csv('./data/book_tags.csv')
```

Etiquetas

```
In [6]: df_tags.iloc[2000:2002]
```

```
Out[6]:
```


	tag_id	tag_name
2000	2000	alex-read
2001	2001	alex-rider

Etiquetas por libro

```
In [7]: df_book_tags.head(3)
```

```
Out[7]:
```

	gr_book_id	tag_id	count
0	1	30574	167697
1	1	11305	37174
2	1	11557	34173

 Incorporar el campo `tag_name` de `df_tags` a `df_book_tags`, sustituyendo el campo `tag_id` por la etiqueta (*String*) correspondiente. Para ello, fundir los dos *DataFrame* y, posteriormente, eliminar las columnas `count` y `tag_id`.

```
In [8]: # Uno los dos DataFrames en función de la columna tag_id
df_merged = df_book_tags.merge(df_tags, on='tag_id')
#display(df_merged)

# Elimino las columnas 'count' y 'tag_id'
df_book_tags = df_merged.drop(['count', 'tag_id'], axis=1)

# Muestro los últimos registros del DataFrame resultante
df_book_tags.tail()
```

```
Out[8]:
```

	gr_book_id	tag_name
999907	31538635	hogwarts
999908	32848471	jan-2017
999909	33288638	single-mom
999910	33288638	fave-author
999911	33288638	slowburn



Inspeccionar las etiquetas que aparecen en `df_book_tags` y el número de veces que aparece cada una. ¿Deberían eliminarse algunas?

```
In [9]: tag_counts = df_book_tags['tag_name'].value_counts()
print(tag_counts)
```

```
to-read          9983
favorites         9881
owned            9858
books-i-own      9799
currently-reading 9776
...
prose-literature    1
best-reads-2016     1
on-the-stack        1
beautiful-pictures  1
slowburn            1
Name: tag_name, Length: 34252, dtype: int64
```

En base a los resultados, parece que hay muchas etiquetas que aparecen solo una vez en el conjunto de datos. Son muy específicas y no demasiado útiles para el análisis, por lo que pueden ser eliminadas para simplificar los datos y reducir el ruido.



Una de las cosas que se observan es que hay un alto número de etiquetas que aparecen una vez, y que son irrelevantes, por lo que es mejor ignorarlas.

Eliminar todas las etiquetas que aparezcan menos de 20 veces.

Este ejercicio se puede hacer de varias formas. Una de ellas, consiste en agrupar por `tag_name`, mediante `groupby`, y filtrar con `filter` los grupos con tamaño  $\geq 20$ .

```
In [10]: # Agrupo por tag_name y cuento el tamaño de cada grupo
tag_counts = df_book_tags.groupby('tag_name').size()

# Filtro Los grupos con tamaño >= 20
popular_tags = tag_counts[tag_counts >= 20].index.tolist()

# Filtro el subconjunto de df_book_tags que contenga solo las etiquetas populares
df_popular_tags = df_book_tags[df_book_tags['tag_name'].isin(popular_tags)]

# Fusiono con df_tags para obtener el nombre completo de la etiqueta
df_popular_tags = df_popular_tags.merge(df_tags, left_on='tag_name', right_on='tag_name')

# Elimino las columnas tag_id
df_popular_tags.drop(['tag_id'], axis=1, inplace=True)
#display(df_popular_tags)

# Cuento el nº de veces que aparece
tag_counts = df_popular_tags['tag_name'].value_counts()
print(tag_counts)
```

```
to-read          9983
favorites        9881
owned            9858
books-i-own      9799
currently-reading 9776
...
early-modern     20
religión         20
genre-historical-fiction 20
genocide         20
kindle-first-books 20
Name: tag_name, Length: 3359, dtype: int64
```



Otro de los problemas que se aprecian es que algunas etiquetas son genéricas, y no corresponden a libros concretos. Por ejemplo palabras como `read-readings` o `favourites` .

```
In [11]: df_book_tags['tag_name'].value_counts().nlargest(25)
```

```
Out[11]: to-read          9983
         favorites        9881
         owned           9858
         books-i-own      9799
         currently-reading 9776
         library          9415
         owned-books      9221
         fiction           9097
         to-buy           8692
         kindle           8316
         default          8239
         ebook            8054
         my-books         7561
         audiobook        7242
         ebooks          7203
         wish-list        7192
         my-library       7000
         audiobooks       6862
         i-own            6670
         adult            6604
         audio            6548
         favourites       6422
         novels           5665
         own-it           5514
         contemporary     5287
         Name: tag_name, dtype: int64
```

 Eliminar todas las etiquetas que contengan estos términos (los términos de la lista `target_tags` ).

❗ En este caso, es recomendable no utilizar las etiquetas completas para eliminar así sus variantes también. Por otra parte, `Series.str.contains` acepta expresiones regulares.



In [12]: `import re`

```
target_tags = ['read', 'favo', 'own', 'top', 'book', 'librar', 'kindle', 'list']

# Creo una expresión regular que busca las palabras que empiezan o contienen los términos de la lista target_tags
pattern = '|'.join([r'\b{}\w*'.format(term) for term in target_tags])

# Creo una máscara booleana que indica si el tag_name contiene alguna de las palabras buscadas
mask = df_book_tags['tag_name'].str.contains(pattern, flags=re.IGNORECASE, regex=True)

# Aplico la máscara para filtrar las etiquetas que no queremos
df_book_tags = df_book_tags[~mask].reset_index(drop=True)
df_book_tags
```


Out[12]:

	gr_book_id	tag_name
0	1	fantasy
1	2	fantasy
2	3	fantasy
3	5	fantasy
4	6	fantasy
...	...	...
671111	31538635	hogwarts
671112	32848471	jan-2017
671113	33288638	single-mom
671114	33288638	fave-author
671115	33288638	slowburn

671116 rows × 2 columns



Por último, se creará un `DataFrame` denominado `df_book_tag_text` en el que, para cada libro (indicado por su código `goodreads_book_id`), se añadirá una columna con *un solo campo de texto*, resultado de unir las etiquetas correspondientes.

 Agrupar las entradas `DataFrame` en `df_book_tags` en función del campo `goodreads_book_id` y unir todas las etiquetas de cada grupo mediante `join`. Almacenar el resultado en un `DataFrame` denominado `df_book_tag_text`.


 Para pasos posteriores es necesario convertir el resultado resultante, una `Serie`, en un `DataFrame`, mediante `Series.to_frame()`.

```
In [13]: df_book_tag_text = df_book_tags.groupby('gr_book_id')['tag_name'].apply(lambda x: ' '.join(x)).to_frame()
df_book_tag_text.head()
```

Out[13]:

	tag_name
gr_book_id	
1	fantasy young-adult fiction harry-potter ya se...
2	fantasy young-adult fiction harry-potter ya ma...
3	fantasy young-adult fiction harry-potter ya se...
5	fantasy young-adult fiction harry-potter ya se...
6	fantasy young-adult fiction harry-potter ya se...




 Renombrar la columna `tag_name` de `df_book_tags` a `text_tags` . Rellenar los valores perdidos con un String vacío ("").

```
In [14]: df_book_tag_text = df_book_tag_text.rename(columns={'tag_name': 'text_tags'})
df_book_tag_text['text_tags'] = df_book_tag_text['text_tags'].fillna('')
df_book_tag_text.head()
```

Out[14]:

	text_tags
gr_book_id	
1	fantasy young-adult fiction harry-potter ya se...
2	fantasy young-adult fiction harry-potter ya ma...
3	fantasy young-adult fiction harry-potter ya se...
5	fantasy young-adult fiction harry-potter ya se...
6	fantasy young-adult fiction harry-potter ya se...



 Incorporar la información del texto de las etiquetas, es decir, la columna `text_tags` , al `DataFrame` principal `df_goodreads` (utilizar *merge*).

```
In [15]: df_goodreads = pd.merge(df_goodreads, df_book_tag_text, on='gr_book_id', how='left')
df_goodreads.head(2)
```

```
Out[15]:
```

	gr_book_id	gr_best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	title	...	work_text_reviews_count	ratings_1	ratings_2
0	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	The Hunger Games (The Hunger Games, #1)	...	155254	66715	127936
1	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	Harry Potter and the Sorcerer's Stone (Harry P...	...	75867	75504	101676

2 rows × 24 columns

☒

## 2. Limpieza y preparación del texto

En este proyecto, el contenido (texto) asociado a cada libro se representa mediante un modelo de bolsa de palabras. Como se ha explicado a lo largo del módulo, cuando se trabaja con este tipo de representación es recomendable limpiar el texto y eliminar información irrelevante. Para ello se utilizará la librería `spacy`. En primer lugar, se utilizará el modelo de lenguaje `en_core_web_sm`. Aunque no es el modelo con mejor rendimiento, es suficiente para este contexto, y es más eficiente que `en_core_web_trf` (basado en *transformers*, y que es el que mejor funciona).

```
In [16]: #!pip install spacy
```

```
In [17]: #! python -m spacy download es_dep_news_trf
#! python -m spacy download en_core_web_sm # Solo se utilizará este
#! python -m spacy download en_core_web_trf
```

```
In [18]: #! python -m spacy download en_core_web_sm
```

In [19]: `import spacy`

```
# Creo un objeto con el pipeline
nlp = spacy.load("en_core_web_sm")
nlp.pipe_names
```

Out[19]: ['tok2vec', 'tagger', 'parser', 'attribute\_ruler', 'lemmatizer', 'ner']

✍ Crear una función denominada `clean` que acepte un texto, lo convierta en un documento ( `spacy` ) y devuelva un *String* compuesto por los lemas correspondientes a cada token, en minúscula, y descartando los tokens que no sean alfanuméricos, o los que correspondan a *stopwords*.

```
In [20]: def clean(overview):
# Proceso el texto con el objeto spacy
doc = nlp(overview)

# Lematizo los tokens restantes, eliminado los que no son alfanuméricos o stopwords
lemmas = [token.lemma_.lower() for token in doc if token.is_alpha and not token.is_stop]

# Uno los lemas en un String separados por un espacio
clean_text = ' '.join(lemmas)

return clean_text

overview = df_goodreads.iloc[0]['overview']
clean(overview)
```

Out[20]: 'win famous lose mean certain death nation panem form post apocalyptic north america country consist wealthy capitol region surround poor district early hi story rebellion lead district capitol result destruction creation annual televise event know hunger games punishment reminder power grace capitol district yield boy girl age lottery system participate game tribute choose annual reaping force fight death leave survivor claim victory year old katniss young sist er prim select district female representative katniss volunteer place male counterpart peeta pit big strong representative train life see death sentence ka tniss close death survival second nature'



✍ Aplicar la función `clean` sobre cada texto almacenado en la columna `df_goodreads['overview']` para preprocesarlo, almacenando el resultado en la propia columna.

⚠ Este ejercicio tarda unos cinco minutos en ejecutarse. Se ha planteado así (sin utilizar `nlp.pipe` ) porque es más sencillo, y en este caso tarda prácticamente lo mismo.

```
In [21]: df_goodreads['overview'] = df_goodreads['overview'].apply(clean)
```


```
In [22]: print(df_goodreads['overview'])
```

```
0      win famous lose mean certain death nation pane...
1      harry potter life miserable parent dead stuck ...
2      thing absolutely positive edward vampire secon...
3      unforgettable novel childhood sleepy southern ...
4      publication great gatsby largely dismiss light...
...
9995   edge lie world border broken people shop walma...
9996   robert caro life lyndon johnson begin greatly ...
9997   aubrey maturin volume actually constitute sing...
9998   acclaimed author groundbreaking bestseller sch...
9999   world war create modern world conflict unprece...
Name: overview, Length: 10000, dtype: object
```



## Preparación de datos

Una vez incorporada la información relativa a resúmenes y etiquetas, y puesto que el texto se tratará simplemente como una bolsa de palabras, esta se fundirá, constituyendo el contenido de cada libro.

 Fundir las columnas `overview` y `text_tags` de `df_goodreads` en otra columna llamada `text`. Eliminar las columnas en las que `text` tenga longitud 0.

```
In [23]: df_goodreads['text'] = df_goodreads['overview'].str.strip() + ' ' + df_goodreads['text_tags'].str.strip()
display(df_goodreads['text'])
```

```
df_goodreads.shape
```

```
0      win famous lose mean certain death nation pane...
1      harry potter life miserable parent dead stuck ...
2      thing absolutely positive edward vampire secon...
3      unforgettable novel childhood sleepy southern ...
4      publication great gatsby largely dismiss light...
...
9995   edge lie world border broken people shop walma...
9996   robert caro life lyndon johnson begin greatly ...
9997   aubrey maturin volume actually constitute sing...
9998   acclaimed author groundbreaking bestseller sch...
9999   world war create modern world conflict unprece...
Name: text, Length: 10000, dtype: object
```

```
Out[23]: (10000, 25)
```

```
In [24]: data_books = df_goodreads['text']
```



### Selección de cinco libros para pruebas

A continuación, se eligen cinco libros, que serán almacenados en el DataFrame `df_books_test`

ⓘ Como no vamos a hacer predicción, no pasa nada porque los libros estén en dos sitios.

```
In [25]: test_pos = np.array([2,4,7,9,12]) # Sugerencia. Podéis cambiarlos.

df_books_test = df_goodreads.iloc[test_pos].copy()
df_books_test
```


Out[25]:

	gr_book_id	gr_best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	title	...	ratings_1	ratings_2	ratings_3	ratings_4	rating
2	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0	Twilight	Twilight (Twilight, #1)	...	456191	436802	793319	875073	135
4	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0	The Great Gatsby	The Great Gatsby	...	86236	197621	606158	936012	94
7	5107	5107	3036731	360	316769177	9.780317e+12	J.D. Salinger	1951.0	The Catcher in the Rye	The Catcher in the Rye	...	109383	185520	455042	661516	70
9	1885	1885	3060926	3455	679783261	9.780680e+12	Jane Austen	1813.0	Pride and Prejudice	Pride and Prejudice	...	54700	86485	284852	609755	115
12	5470	5470	153313	995	451524934	9.780452e+12	George Orwell, Erich Fromm, Celâl Üster	1949.0	Nineteen Eighty-Four	1984	...	41845	86425	324874	692021	90

5 rows × 25 columns

### 3. Búsqueda por similaridad de la representación tf-idf

La información *tf-idf* es muy útil de cara a clasificar y comparar documentos. Cada texto se representa mediante un vector de valores *tf-idf*, y la búsqueda de documentos se apoya en una medida de similaridad, la *similaridad coseno*, para esta representación.

 Crear un objeto `TfidfVectorizer` de `sklearn`, denominado `tfidf_vect`, que represente un máximo de 20000 términos. Obtener la matriz *tf-idf* de los datos almacenados en `data_books` y almacenarla en una variable denominada `data_books_tfidf`. Extraer los términos considerados en `tfidf_vect`, y almacenarlos en la variable `terms`.

```
In [26]: from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(max_features=20000)
data_books_tfidf = tfidf_vect.fit_transform(df_goodreads['text'])
terms = tfidf_vect.get_feature_names_out()
```



### Búsqueda del libro más similar

Se tomará un libro incluido en el conjunto de datos `df_books_test`. Por ejemplo, el primero.




```
In [27]: print(df_books_test.iloc[0])
text_query = df_books_test.iloc[0].text
text_query
```

```
gr_book_id          41865
gr_best_book_id     41865
work_id             3212258
books_count         226
isbn                316015849
isbn13              9780316015840.0
authors             Stephenie Meyer
original_publication_year  2005.0
original_title       Twilight
title                Twilight (Twilight, #1)
language_code       en-US
average_rating       3.57
ratings_count        3866839
work_ratings_count   3916824
work_text_reviews_count  95009
ratings_1            456191
ratings_2            436802
ratings_3            793319
ratings_4            875073
ratings_5            1355439
image_url            https://images.gr-assets.com/books/1361039443m... (https://images.gr-assets.com/books/1361039443m...)
small_image_url      https://images.gr-assets.com/books/1361039443s... (https://images.gr-assets.com/books/1361039443s...)
overview             thing absolutely positive edward vampire secon...
text_tags             fantasy young-adult fiction ya sci-fi-fantasy ...
text                 thing absolutely positive edward vampire secon...
Name: 2, dtype: object
```

```
Out[27]: 'thing absolutely positive edward vampire second know dominant thirst blood unconditionally irrevocably love book twilight saga internationally bestselle a
uthor stephenie meyer introduce bella swan edward cullen pair star cross lover forbidden relationship ripen backdrop small town suspicion mysterious coven
vampire love story bite fantasy young-adult fiction ya sci-fi-fantasy default reread audiobook 5-stars novels fantasy-sci-fi audiobooks paranormal ya-fanta
sy teen english urban-fantasy supernatural ya-fiction young-adult-fiction scifi-fantasy faves ebook contemporary on-my-shelf have movie finished sci-fi ya-
lit werewolves séries love science-fiction finished-series youngadult abandoned did-not-finish dnf american movies drama high-school romantic horror chick-
lit first-in-series teen-fiction meh young-adults guilty-pleasures guilty-pleasure paranormal-romance ya-paranormal ya-romance vampires vampire completed-s
eries never-again vamps paranormal-fantasy fantasy-paranormal pnr love-triangle twilight stephenie-meyer twilight-saga twilight-series stephanie-meyer'
```

Para poder buscar otro libro por similaridad, es necesario transformar el texto de consulta a la representación *tf-idf*. Previamente, ha de ser preprocesado (limpiado con la función `clean` ).

 Transformar el documento `text_query` a formato *tf-idf* y almacenar el resultado en `text_query_tfidf` .

❗ Esta sería la manera de proceder con libros nuevos. No obstante, en este caso concreto, **como el libro ya estaba en el corpus y se ha obtenido su representación**, se podría obtener el vector directamente desde `data_books_tfidf` .

```
In [28]: text_query_tfidf = tfidf_vect.transform([clean(text_query)])
```



Mostrar los 10 términos más relevantes (con mayor valor *tf-idf*) para el documento `text_query`.

```
In [29]: top_indices = np.argsort(text_query_tfidf.toarray()[::-1][:, :10])

for i, top_idx in enumerate(top_indices):
    top_terms = [terms[idx] for idx in top_idx]
    print(f"Top 10 términos del documento {i+1}: {top_terms}")
```

Top 10 términos del documento 1: ['twilight', 'meyer', 'vampire', 'fantasy', 'stephenie', 'ya', 'paranormal', 'edward', 'saga', 'ripen']



Calcular las similitudes (similitud coseno) del documento `text_query_tfidf` con el resto de documentos en la matriz `data_books_tfidf`, y almacenarlas en un vector denominado `similarities_tfidf`.

La función `cosine_similarity` toma como argumentos dos matrices y devuelve una matriz (aunque sea unidimensional). Para obtener un vector unidimensional, y operar posteriormente con más agilidad, `similarities_tfidf` ha de almacenar el elemento cero de la matriz devuelta por la función, o transformar esta matriz en un vector.

```
In [30]: from sklearn.metrics.pairwise import cosine_similarity

similarities_tfidf = cosine_similarity(text_query_tfidf, data_books_tfidf)[0]
```

Obtener los índices de los 6 libros más similares a `text_query` (obviamente, el más similar será el propio libro).

```
# Obtener los índices de los 6 libros más similares a text_query
top_similarities = np.argsort(similarities_tfidf[::-1][1:7])#pongo 1:7 para excluir a si mismo

# Mostrar los títulos de los libros más similares
print(df_goodreads.iloc[top_similarities]['title'])
```

```
In [31]: # Obtengo Las similitudes coseno
similarities_tfidf = cosine_similarity(text_query_tfidf, data_books_tfidf)

# Obtengo el índice del libro más similar
most_similar_index = np.argmax(similarities_tfidf)

# Obtengo Las similitudes en orden descendente
sorted_similarities = np.argsort(similarities_tfidf[0])[:-1]

# Selecciono Los 6 índices con Los valores más altos (excluyendo el índice del propio libro)
top_similarities = [i for i in sorted_similarities if i != most_similar_index][:6]

# Muestro Los títulos de Los Libros más similares
df_goodreads.iloc[top_similarities]['title']
```

```
Out[31]: 51                      Eclipse (Twilight, #3)
         48                      New Moon (Twilight, #2)
         991          The Twilight Saga (Twilight, #1-4)
         833          Midnight Sun (Twilight, #1.5)
        1618  The Twilight Saga Complete Collection (Twilig...
        2020          The Twilight Collection (Twilight, #1-3)
         Name: title, dtype: object
```



Se aprecia que, efectivamente, los libros son muy similares. De hecho, todos corresponden a la saga. Esto puede deberse al uso de palabras clave muy concretas como *"Twilight"*.



## 4. Recomendación basada en contenido

Existen muchas posibilidades en la implementación de sistemas de recomendación basados en contenido. En general, todas requieren:

- Determinar qué ítems prefiere el usuario
- Encontrar ítems similares
- Priorizar esos ítems

Para ilustrar el funcionamiento de este tipo de sistemas, se tomarán los datos relativos a la actividad de un usuario escogido al azar, y se devolverá un conjunto de libros recomendados.

El archivo `./data/ratings.csv` contiene las valoraciones hechas por más de 53000 usuarios a los 10000 libros. En total, contiene cerca de un millón de entradas en formato ( `user_id` , `book_id` , `rating` ). El tamaño de la base de datos dificulta el trabajo con una matriz, aunque sea dispersa, ya que el proceso de elaboración (mediante `pivot` es muy lento).

```
In [32]: df_ratings = pd.read_csv('./data/ratings.csv', sep=',')

display(df_ratings.head())
df_ratings.shape
```


	user_id	book_id	rating
0	313	0	5
1	438	0	3
2	587	0	5
3	1168	0	4
4	1184	0	4

```
Out[32]: (981756, 3)
```



## Selección de libros preferidos por el usuario

El primer paso de la recomendación consiste en determinar qué libros prefiere el usuario. Una posibilidad consiste en seleccionar aquellos para los que éste ha otorgado una puntuación mayor que tres.

 Tomar un usuario al azar y devolver las entradas correspondientes a los libros que ha votado. Almacenarlas en el `DataFrame ratings_user`. Descartar de `ratings_user` todos los libros con puntuación menor que cuatro.

```
In [33]: np.random.seed(0)
test_user = np.random.randint(max(df_ratings['user_id']))

ratings_user = df_ratings[(df_ratings['user_id'] == test_user) & (df_ratings['rating'] >= 4)]
ratings_user
```


```
Out[33]:
```

	user_id	book_id	rating	
	53101	2732	531	5
	209776	2732	2099	5
	214080	2732	2142	5
	242344	2732	2425	4
	369792	2732	3702	4
	401262	2732	4018	4
	425738	2732	4264	5
	476351	2732	4773	4
	524632	2732	5262	5
	543609	2732	5454	4
	602404	2732	6048	5
	627124	2732	6298	4
	629500	2732	6322	5
	755541	2732	7610	5



## Búsqueda de libros similares

Una vez se dispone de la lista de libros, es necesario encontrar libros similares.

 Extraer la representación de los libros relativos al usuario y copiarla en una matriz denominada `user_books_tfidf`.

```
In [34]: user_books_tfidf = data_books_tfidf[ratings_user.loc[ratings_user['user_id'] == test_user]['book_id'].values]
user_books_tfidf.shape
```

```
Out[34]: (14, 20000)
```



✍ Calcular la matriz de similitudes coseno entre `user_books_tfidf` , `data_books_tfidf` . Fijar las similitudes de los libros leídos con el usuario ( `books_user_id` ) consigo mismos a -1 con el fin de descartarlos.

```
In [35]: books_user_id = df_ratings[df_ratings['user_id'] == test_user]['book_id'].unique()

similarities_user = cosine_similarity(user_books_tfidf, data_books_tfidf)
similarities_user[:, books_user_id] = -1 # Excluimos Los libros ya leídos

similarities_user.shape
```

Out[35]: (14, 10000)



✍ Obtener los 3 libros más similares *a cada uno* de los leídos por el usuario y almacenarlos en la matriz `similar_books` .

❗ La función `similarities_user.argsort(axis=1)` devuelve los índices ordenados (ascendentemente) para cada fila del array. Se trata de coger las 3 últimas columnas de `similarities_user` .

Por otra parte, el resultado de este ejercicio ha de ser un array *unidimensional* sin elementos repetidos.

```
In [36]: similar_books = []
for row in similarities_user.argsort(axis=1):
    row_similar_books = row[-3:] # Los 3 libros más similares
    row_similar_books = row_similar_books[~np.in1d(row_similar_books, books_user_id)] # Excluimos los libros ya leídos
    similar_books.extend(row_similar_books)

similar_books = np.unique(similar_books)

rec_books_user_id = df_goodreads.loc[similar_books].index.values

print("\033[1mLos libros recomendados son:\n\033[0m")
for book_id in rec_books_user_id:
    title = df_goodreads.loc[book_id]['title']
    rating = df_goodreads.loc[book_id]['average_rating']
    print(f"{book_id}: {title} \u223C (\u2605 {rating})")

print(f"\n\033[1mTotal de libros recomendados: {len(similar_books)}")
```

**Los libros recomendados son:**

147: Girl with a Pearl Earring ~ (★ 3.85)  
175: It ~ (★ 4.18)  
304: Pet Sematary ~ (★ 3.91)  
357: Oh, The Places You'll Go! ~ (★ 4.34)  
387: Hatchet (Brian's Saga, #1) ~ (★ 3.68)  
567: Batman: The Dark Knight Returns (The Dark Knight Saga, #1) ~ (★ 4.25)  
846: The Deep End of the Ocean (Cappadora Family, #1) ~ (★ 3.84)  
947: Empire Falls ~ (★ 3.91)  
1074: Dolores Claiborne ~ (★ 3.81)  
1622: Batman: The Long Halloween ~ (★ 4.29)  
2022: Forrest Gump (Forrest Gump, #1) ~ (★ 4.06)  
2110: Batman: Arkham Asylum - A Serious House on Serious Earth ~ (★ 4.09)  
2474: My Side of the Mountain (Mountain, #1) ~ (★ 4.03)  
2513: Marvel 1602 ~ (★ 3.93)  
2677: Superman: Red Son ~ (★ 4.17)  
3273: Hop On Pop ~ (★ 3.95)  
3321: The Lady and the Unicorn ~ (★ 3.67)  
3559: The Plot Against America ~ (★ 3.71)  
3786: Songs in Ordinary Time ~ (★ 3.68)  
3941: Sin City, Vol. 2: A Dame to Kill For (Sin City, #2) ~ (★ 4.16)  
4172: Civil War: A Marvel Comics Event ~ (★ 4.05)  
4569: The Last Runaway ~ (★ 3.78)  
4937: Ellen Foster ~ (★ 3.75)  
4971: Straight Man ~ (★ 4.02)  
5186: Nobody's Fool ~ (★ 4.12)  
5825: In the Company of Cheerful Ladies (No. 1 Ladies' Detective Agency, #6) ~ (★ 4.07)  
6335: The Good Husband of Zebra Drive (No. 1 Ladies' Detective Agency, #8) ~ (★ 4.06)  
6444: Akira, Vol. 1 ~ (★ 4.36)  
6754: The River (Brian's Saga, #2) ~ (★ 3.77)  
6975: Superman: Earth One, Volume 1 ~ (★ 3.9)  
7087: The Miracle at Speedy Motors (No.1 Ladies' Detective Agency, #9) ~ (★ 4.09)  
7578: Goodbye, Columbus and Five Short Stories ~ (★ 3.87)  
7793: Everyman ~ (★ 3.54)  
8213: Sin City, Vol. 3: The Big Fat Kill (Sin City, #3) ~ (★ 4.1)  
8379: Legally Blonde ~ (★ 3.68)  
9211: Braveheart ~ (★ 4.19)  
9600: Superman: Whatever Happened to the Man of Tomorrow? ~ (★ 4.14)  
9745: In a People House ~ (★ 4.13)  
9778: Iron Man: Extremis ~ (★ 4.08)

**Total de libros recomendados: 39**





## Priorización de resultados y recomendación

Llegados a este punto, se han obtenido los libros más similares a los leídos y valorados positivamente por el usuario. Una posibilidad a la hora de priorizar los resultados consistiría en dar más peso a aquellos libros similares que han aparecido más veces en la matriz `similar_books`.

También es posible utilizar la información del `DataFrame` `df_goodreads` para priorizar los libros. En concreto, la columna `average_rating` contiene la valoración media de cada libro en la plataforma, y puede ser utilizada para priorizar.

🔗 Obtener los 10 libros con mayor `average_rating` de entre los más similares a los valorados positivamente por el usuario ( `similar_books` ). Almacenar sus índices en un array

```
In [37]: rec_books_user_id = df_goodreads.loc[similar_books].sort_values(by='average_rating', ascending=False)[:10].index.values
```

```
print("\033[1mLos 10 libros recomendados con mayor puntuación son:\n\033[0m")
for book_id in rec_books_user_id:
    title = df_goodreads.loc[book_id]['title']
    rating = df_goodreads.loc[book_id]['average_rating']
    print(f"{book_id}: {title} \u223C (\u2605 {rating})")
```

**Los 10 libros recomendados con mayor puntuación son:**

```
6444: Akira, Vol. 1 ~ (★ 4.36)
357: Oh, The Places You'll Go! ~ (★ 4.34)
1622: Batman: The Long Halloween ~ (★ 4.29)
567: Batman: The Dark Knight Returns (The Dark Knight Saga, #1) ~ (★ 4.25)
9211: Braveheart ~ (★ 4.19)
175: It ~ (★ 4.18)
2677: Superman: Red Son ~ (★ 4.17)
3941: Sin City, Vol. 2: A Dame to Kill For (Sin City, #2) ~ (★ 4.16)
9600: Superman: Whatever Happened to the Man of Tomorrow? ~ (★ 4.14)
9745: In a People House ~ (★ 4.13)
```



Por último, se imprimen los nombres de los libros preferidos por el usuario, cuyos índices están almacenados en `books_user_id`, y la recomendación, cuyos índices están almacenados en `rec_books_user_id`.

```
In [38]: print("Libros preferidos por el usuario: ")
print("-----")
for book in df_goodreads.iloc[books_user_id][['title', 'authors']].itertuples():
    print(book.authors, "\n\t",book.title)

print("\n\nRecomendación: ")
print("-----")
for book in df_goodreads.iloc[rec_books_user_id][['title', 'authors']].itertuples():
    print(book.authors, "\n\t",book.title)
```

Libros preferidos por el usuario:

-----

Frank Miller, David Mazzucchelli, Richmond Lewis, Dennis O'Neil

Batman: Year One

Alan Moore, Brian Bolland, Tim Sale

Batman: The Killing Joke

Frank Miller

Batman: The Dark Knight Returns #1

Mark Waid, Alex Ross, Elliot S. Maggin

Kingdom Come

Frank Miller, Lynn Varley

300

Philip Roth

The Human Stain (The American Trilogy, #3)

Kurt Busiek, Alex Ross

Marvels

A. Manette Ansay

Vinegar Hill

Michael Blake

Dances with Wolves (Dances with Wolves, #1)

Dean Koontz

Night Chills

Jim Starlin, Jim Aparo, Mike DeCarlo

Batman: A Death in the Family

Alexander McCall Smith

The Full Cupboard of Life (No. 1 Ladies' Detective Agency, #5)

Scott B. Smith

A Simple Plan

Gary Paulsen

Brian's Winter (Brian's Saga, #3)

Richard Russo

Bridge of Sighs

Tracy Chevalier

Falling Angels

Mary Higgins Clark

Remember Me

Stephen King

Apt Pupil

Jeph Loeb, Jim Lee, Scott Williams

Batman: Hush, Vol. 2

Dr. Seuss

And to Think That I Saw it on Mulberry Street

Jonathan Harr

A Civil Action

Mary Higgins Clark

The Cradle Will Fall

Jeph Loeb, Jim Lee, Scott Williams

Batman: Hush, Vol. 1

Recomendación:

-----

Katsuhiro Otomo, Yoko Umezawa, Linda M. York, Jo Duffy  
Akira, Vol. 1  
Dr. Seuss  
Oh, The Places You'll Go!  
Jeph Loeb, Tim Sale, Gregory Wright, Richard Starkings  
Batman: The Long Halloween  
Frank Miller, Klaus Janson, Lynn Varley  
Batman: The Dark Knight Returns (The Dark Knight Saga, #1)  
Randall Wallace  
Braveheart  
Stephen King  
It  
Mark Millar, Kilian Plunkett, Andrew Robinson, Walden Wong, Dave Johnson  
Superman: Red Son  
Frank Miller  
Sin City, Vol. 2: A Dame to Kill For (Sin City, #2)  
Alan Moore, Curt Swan, George Pérez, Kurt Schaffenberger  
Superman: Whatever Happened to the Man of Tomorrow?  
Dr. Seuss, Theo LeSieg, Roy McKie  
In a People House

Puede observarse que, efectivamente, los títulos recomendados guardan mucha relación con los títulos valorados positivamente por el usuario.



## 5. Sistema híbrido de recomendación

En el apartado anterior se ha utilizado la valoración media de cada libro para priorizar los vecinos más cercanos. Existe otra posibilidad, que consiste en utilizar las valoraciones de los libros obtenidas mediante filtrado colaborativo. Esto constituye un sistema híbrido, ya que se mezclan las dos aproximaciones. Por una parte, se seleccionan libros basados en contenido, y de ellos, se muestra el conjunto para el que se predice una mayor puntuación por parte del usuario.

No existe mucho material disponible en relación a este tipo de modelos. Tampoco una librería de referencia, aunque va cobrando popularidad [surprise \(http://surpriselib.com/\)](http://surpriselib.com/). En primer lugar, se utilizará esta librería para obtener los scores mediante filtrado colaborativo. En concreto, se utilizará el algoritmo SVD (visto en clase).

**i** El autor principal de la librería describe en una serie de artículos [enlace \(http://nicolas-hug.com/blog/\)](http://nicolas-hug.com/blog/) como funciona la versión básica de este algoritmo. La lectura es muy recomendable.


En la siguiente celda, se cargan los votos en la estructura `ratings_data` (usada por `surprise`) y se aprende un modelo SVD.

Para instalar `surprise` hay que seguir estos pasos (da error si se hace desde JupyterLab):

1. En el apartado 'Home' del programa instalado Anaconda Navigator, buscamos 'CMD.exe Prompt' y clickamos en 'Launch'.
2. En la terminal escribimos: conda update --all
3. Una vez ejecutada la actualización, escribimos en la terminal: conda install -c conda-forge scikit-surprise

```
In [39]: from surprise import SVD, Dataset, Reader
from surprise.model_selection.split import train_test_split

reader = Reader()
ratings_data = Dataset.load_from_df(df_ratings, reader).build_full_trainset()
svd = SVD(n_factors=10)
svd.fit(ratings_data);
```

 Obtener uno a uno, las valoraciones de los libros similares a los preferidos por el usuario ( similar\_books ), y tomar los 10 con más valoración.

```
In [40]: scores = np.zeros(similar_books.shape)

# Obtengo Las valoraciones de Los libros similares
for i, book_id in enumerate(similar_books):
    scores[i] = svd.predict(reader, book_id).est

# Obtengo Los índices de Los 10 libros con mayor valoración
top_10_indices = np.argsort(scores)[::-1][:10]

# Obtengo Los IDs de Los 10 libros con mayor valoración
top_10_book_ids = similar_books[top_10_indices]
display(top_10_book_ids)

#Imprimo Las recomendaciones
print("\n\nRecomendación: ")
print("-----")
for book in df_goodreads.iloc[top_10_book_ids][['title', 'authors']].itertuples():
    print(book.authors, "\n\t", book.title)
```

```
array([ 567, 6444, 1622, 5186, 3941, 9600,  357, 2474, 4971, 2110],
      dtype=int64)
```

Recomendación:

-----

```
Frank Miller, Klaus Janson, Lynn Varley
    Batman: The Dark Knight Returns (The Dark Knight Saga, #1)
Katsuhiro Otomo, Yoko Umezawa, Linda M. York, Jo Duffy
    Akira, Vol. 1
Jeph Loeb, Tim Sale, Gregory Wright, Richard Starkings
    Batman: The Long Halloween
Richard Russo
    Nobody's Fool
Frank Miller
    Sin City, Vol. 2: A Dame to Kill For (Sin City, #2)
Alan Moore, Curt Swan, George Pérez, Kurt Schaffenberger
    Superman: Whatever Happened to the Man of Tomorrow?
Dr. Seuss
    Oh, The Places You'll Go!
Jean Craighead George
    My Side of the Mountain (Mountain, #1)
Richard Russo
    Straight Man
Grant Morrison, Dave McKean
    Batman: Arkham Asylum - A Serious House on Serious Earth
```



## 6. LDA y búsqueda por similitud

LDA permite modelar los grupos que dan lugar al corpus de documentos, de modo que cada documento se representa como una mezcla de las distribuciones de probabilidad de cada grupo, dada por un vector de pesos. Esta representación puede ser utilizada también para caracterizar el documento (en lugar de *tf-idf*) y permite cuantificar la similitud con otros. Por tanto, también serviría para implementar un sistema de recomendación (basado en contenido o híbrido) similar al anterior.

LDA trabaja con la matriz de frecuencias de términos por documento (*term frequency*). Por tanto, es necesario primero obtenerla.

✍ Construir un objeto `CountVectorizer` de `sklearn` y almacenarlo en la variable `cv_vect`. Obtener la matriz `data_books_cv` a partir de los datos almacenados en `data_books`. Utilizar un tamaño máximo de vocabulario de 5000 (para que LDA no requiera demasiado tiempo y memoria). Almacenar el vocabulario generado en `cv_vect` en la variable `terms`.

```
In [41]: from sklearn.feature_extraction.text import CountVectorizer
```

```
c_vect = CountVectorizer(max_features=5000)
data_books_cv = c_vect.fit_transform(data_books)
terms = c_vect.get_feature_names_out()
```



Una vez construida la matriz de frecuencias, se ejecutará el algoritmo LDA, para representar los documentos mediante 20 grupos.

```
In [42]: from sklearn.decomposition import LatentDirichletAllocation
```

```
# Número de temas
n_topics = 20

lda = LatentDirichletAllocation(n_components=n_topics, max_iter=20, learning_method='online',
                               learning_offset=50., random_state=0, n_jobs=-1)
```

✍ Construir el modelo LDA y almacenarlo en la variable `topic_term`. Guardar la representación de los documentos en la variable `data_books_lda`.

```
In [43]: from time import time
start = time()

data_books_lda = lda.fit_transform(data_books_cv)
topic_term = lda.components_

print("Tiempo: {:.3f}s.".format(time() - start))
print("Tamaño del modelo:", topic_term.shape)
print("Tamaño del resultado de la transformación: ", data_books_lda.shape)
```

```
Tiempo: 113.635s.
Tamaño del modelo: (20, 5000)
Tamaño del resultado de la transformación: (10000, 20)
```



La siguiente función (de la práctica de LDA) imprime los términos y scores más importantes para cada grupo.

```
In [44]: def topic_relevant_words(topic_id, model, terms, num_words):
    print("Grupo: ", topic_id)
    print("-----")
    # Extrae las posiciones de los mayores scores.
    top_term_ids = model[topic_id,:].argsort()[::-num_words-1:-1]
    # Extrae los términos de las posiciones correspondientes
    top_terms = [terms[id_term] for id_term in top_term_ids]
    # Extrae y calcula los scores
    top_scores = model[topic_id,top_term_ids] / np.sum(model[topic_id,:])
    # Imprime los resultados
    for term, score in zip(top_terms,top_scores):
        print("{:s}:      \t{:.4f}".format(term,score))
    print("-----\n")
```

```
topic_relevant_words(0, topic_term, terms, 10)
```

```
Grupo:  0
-----
graphic:      0.1259
comics:       0.0949
manga:        0.0903
novels:       0.0727
novel:        0.0377
comic:        0.0331
and:          0.0329
dc:           0.0106
art:          0.0103
series:       0.0091
-----
```

### Búsqueda del libro más similar

Obtener la representación a partir del modelo `lda` del documento `text_query`. Almacenarla en la variable `text_query_lda`.

```
In [45]: text_query_lda = lda.transform(c_vect.transform([text_query]))
text_query_lda
```

```
Out[45]: array([[0.00034965, 0.00034965, 0.00034965, 0.00034965, 0.25626484,
                  0.00034965, 0.00034965, 0.00034965, 0.3216501 , 0.00034965,
                  0.00034965, 0.00034965, 0.00034965, 0.08407762, 0.09314316,
                  0.19276345, 0.00034965, 0.00034965, 0.00034965, 0.04720573]])
```





 Mostrar las características de los tres grupos más relevantes en el texto `text_query` .

```
In [46]: # Encuentro Los índices de Los 3 grupos más relevantes en text_query_lda
top_topic_ids = text_query_lda[0].argsort()[::-4::-1]

# Imprimo Las características para cada uno de estos grupos
for topic_id in top_topic_ids:
    topic_relevant_words(topic_id, topic_term, terms, 10)
```

Grupo: 8

```
-----
fantasy:      0.0613
paranormal:   0.0588
series:       0.0401
romance:      0.0269
urban:        0.0202
supernatural: 0.0190
adult:        0.0172
vampire:      0.0160
ya:           0.0158
stars:        0.0134
-----
```

Grupo: 4

```
-----
fantasy:      0.1290
fiction:      0.0503
sci:          0.0476
fi:           0.0468
science:      0.0380
scifi:        0.0277
sf:           0.0181
and:          0.0155
adult:        0.0141
series:       0.0139
-----
```

Grupo: 15

```
-----
fiction:      0.0433
ya:           0.0326
adult:        0.0229
young:        0.0224
contemporary: 0.0219
love:         0.0179
to:           0.0162
school:       0.0153
buy:          0.0144
romance:      0.0138
-----
```



🔗 Calcular las similitudes (distancia\_euclídea) del documento `text_query_lda` con el resto de documentos en la matriz `data_books_lda` , y almacenarlas en un vector denominado `similarities_lda` .

```
In [47]: from sklearn.metrics.pairwise import euclidean_distances

similarities_lda = euclidean_distances(text_query_lda, data_books_lda)
```



🔗 Obtener los índices de los 10 libros más similares a `text_query` . Hay que tener en cuenta que `euclidean_distances` devuelve distancias, no similitudes, por lo que hay que tomar los primeros índices.

```
In [48]: top_similarities = similarities_lda.argsort()[0][:10]

df_goodreads.iloc[top_similarities]['title']
```

```
Out[48]: 2          Twilight (Twilight, #1)
48          New Moon (Twilight, #2)
991      The Twilight Saga (Twilight, #1-4)
55          Breaking Dawn (Twilight, #4)
51          Eclipse (Twilight, #3)
6732         Peeps (Peeps, #1)
833      Midnight Sun (Twilight, #1.5)
731  The Short Second Life of Bree Tanner: An Eclip...
5809         The Secret Hour (Midnighters, #1)
8636    Touching Darkness (Midnighters, #2)
Name: title, dtype: object
```



En este caso las recomendaciones no son tan evidentes. Sin embargo, se aprecia que la temática es muy parecida a la de "*Twilight*".



