گزارش 6

پیاده‌سازی یک سیستم توصیه گر برای یک مسئله کاربردی

نام و نام خانوادگی : مریم علیپور

شماره دانشجویی : 96120237

استاد : دکتر مهدی قطعی

معرفی دیتاست

مجموعه داده‌های خوبی برای فیلم‌ها (Movielens ، Netflix) و موسیقی (Million songs) وجود دارد، اما نه برای کتاب‌ها.
این مجموعه داده شامل ده هزار کتاب محبوب است. در مورد منبع، بگذارید بگوییم که این رتبه بندی‌ها در اینترنت پیدا شده است. به طور کلی، برای هر کتاب 100 ریویو وجود دارد، اگرچه برخی از آن‌ها دارای رتبه بندی کمتر هستند. رتبه بندی ها از یک به پنج می‌رسد.
book IDها و user IDها در یک رنج هستند. برای کتاب‌ها ، آنها 1-10000، برای کاربران، 1-53424 هستند. همه کاربران حداقل دو رتبه بندی کرده‌اند. تعداد متوسط رتبه بندی برای هر کاربر 8 است.
همچنین کتاب‌هایی برای خواندن توسط کاربران، نشانه‌گذاری شده است.
books.csv برای هر کتاب متا دیتا دارد (average rating، title، authors، goodreads IDs و غیره).

سیستم توصیه‌گر چیست؟

سیستم توصیه‌گر یک زیر کلاس از سیستم فیلتر کردن اطلاعات است که به دنبال پیش بینی رتبه/اولویت کاربر برای یک مورد است.

آن‌ها در درجه اول در برنامه‌هایی که شخص/نهادی با یک محصول/خدمات درگیر هستند استفاده می‌شود. برای بهبود بیشتر تجربه آن‌ها با این محصول، سعی می‌کنیم آن را متناسب با نیازهای آن‌ها شخصی سازی کنیم. برای این منظور ما باید در تعاملات گذشته آنها با این محصول جستجو کنیم.

انواع سیستم‌های توصیه‌گر:

- Popularity Based
- Classification Based
- Content Based
- Collaborative Based
- Hybrid Based (Content + Collaborative)
- Association Based Rule Mining

سیستم توصیه‌گر مبتنی بر محتوا:

محتوا را بر اساس توضیحات محصول پیشنهاد می‌کند. در اینجا ما می‌خواهیم کتاب‌هایی را فقط بر اساس عنوان کتاب پیشنهاد کنیم. کتاب‌های مشابه ممکن است دارای نام‌های مشابهی باشند و از این رو که دارای تشابه کسینوس بالا هستند.

برای مثال یک سیستم توصیه‌گر می‌تواند پیشنهاد فیلم مبتنی بر محتوا کند که با استفاده از NLP به عنوان مجموعه داده فیلم دارای ویژگی‌های بسیاری مانند توصیف فیلم، کارگردان، بازیگر است که به توصیه بهتر یک فیلم کمک می‌کند.

در این گزارش می‌خواهیم یک سیستم توصیه‌گر برای کتاب پیاده‌سازی کنیم.

ایمپورت کردن پکیج‌ها و دیتاست‌ها:

[1]:
```
import numpy as np
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer

data = pd.read_csv('data/books.csv')
data.head()
```

[1]:

	id	book_id	best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	...	ratings_count	work_ratings_count	work_text_reviews_count	ratings_1	ratings_
0	1	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	...	4780653	4942365	155254	66715	12793
1	2	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	...	4602479	4800065	75867	75504	10167
2	3	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0	Twilight	...	3866839	3916824	95009	456191	43680
3	4	2657	2657	3275794	487	61120081	9.780061e+12	Harper Lee	1960.0	To Kill a Mockingbird	...	3198671	3340896	72586	60427	11741
4	5	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0	The Great Gatsby	...	2683664	2773745	51992	86236	19762

5 rows × 23 columns

پردازش داده:

کارهایی که باید انجام دهیم:

- فقط ستون‌های مربوطه را استخراج می‌کنیم (ستون هایی که متن مورد نیاز ماست)
- بررسی می‌کنیم که آیا مقادیر null در ستون‌های استخراج شده وجود دارد یا خیر.

[4]:
```
#Extract relevant columns that would influence a book's rating based on book title.
books_title = data[['book_id', 'title']]
books_title.head()
```

[4]:

	book_id	title
0	2767052	The Hunger Games (The Hunger Games, #1)
1	3	Harry Potter and the Sorcerer's Stone (Harry P...
2	41865	Twilight (Twilight, #1)
3	2657	To Kill a Mockingbird
4	4671	The Great Gatsby

برای توصیه کتاب مبتنی بر محتوا، ما باید از تکنیک‌های NLP استفاده کنیم:

- استخراج کلمات کلیدی (استخراج کلمات کلیدی از عنوان‌ها)
- تشابه کسینوسی (پیدا کردن تشابه کسینوسی بین همه‌ی عناوین کتاب‌ها)

استخراج کلمات کلیدی:

استخراج کلید واژه تشخیص خودکار اصطلاحاتی است که موضوع یک متن را به بهترین وجه توصیف می‌کند.
برای استخراج کلمات کلیدی ما بر اساس نیاز خود از یکی از موارد زیر استفاده می‌کنیم:

- **CountVectorizer** - روشی ساده برای توکن گذاری کردن مجموعه‌ای از اسناد متنی و ساخت دایره لغات واژه‌های شناخته شده و رمزگذاری اسناد جدید با استفاده از آن دایره لغات.
- **Tf-Idf Vectorizer** - تعداد دفعاتی که یک کلمه در یک متن ظاهر می‌شود را پیدا می کند و سپس این تعداد را با تعداد بارهایی که این کلمه در یک دسته از اسناد دیگر مجموعه نمایش داده شده است را مقایسه می‌کند. سپس برای هر کلمه مهم برای یک متن رنک مشخص می‌کند اگر آن کلمه تعداد دفعات زیادی در آن متن ظاهر شده باشد اما در متن‌های دیگر نشده باشد.
- **Rake** - از Rake زمانی استفاده می‌شود که بخواهید کلمات کلیدی را بدون هیچ زمینه خاصی استخراج کنید (اگرچه از مجموعه متداول کلمات متوقف استفاده می‌کند)

در این گزارش ما از CountVectorizer استفاده می‌کنیم چون می‌خواهیم کلمات کلیدی را فقط از عناوین کتاب استخراج کنیم.

Count Vectorizer:

مجموعه‌ای از اسناد متنی را به ماتریسی از تعداد توکن تبدیل می‌کنیم. این یک جدول داده است که پس از نرمال‌سازی داده‌های توالی نسل بعدی بدست می‌آید.

Data = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', ' the', 'lazy', 'dog']

	The	quick	brown	fox	jumps	over	lazy	dog
Data	2	1	1	1	1	1	1	1

کارهایی که باید انجام دهیم:

- مقداردهی اولیه و متناسب کردن CountVectorizer در عنوان -> برای ایجاد count_matrix این برای شباهت کسینوس مفید است.
- تمام کلمات / ویژگی‌های دایره لغات را بررسی می‌کنیم.
- شباهت کسینوسی را جنریت می‌کنیم.

```
[5]: #Lets vectorize all these titles
from sklearn.feature_extraction.text import CountVectorizer

#initialize vectorizer
vect = CountVectorizer(analyzer = 'word', ngram_range = (1,2), stop_words = 'english', min_df = 0.002) #min_df = rare words, max_df = most used words
#ngram_range = (1,2) - if used more than  1(value), lots of features or noise

#Fit into the title
vect.fit(books_title['title'])
title_matrix = vect.transform(books_title['title'])
title_matrix.shape

[5]: (10000, 261)
```

تعداد کل ویژگی‌هایی که می‌توانیم استخراج کنیم به دلیل آستانه min_df که اعمال کردیم 261 است.
چه اتفاقی می افتد که مقدار min_df را تغییر می‌دهیم؟ این استدلال چه چیزی را کنترل می‌کند؟
هنگام ساخت دایره لغات، اصطلاحاتی را که فرکانس سند آن‌ها کاملاً کمتر از حد مجاز است، نادیده گرفته می‌شود. در ادبیات به این مقدار برش نیز گفته می‌شود. اگر float باشد، پارامتر نمایانگر نسبت اسناد، تعداد مطلق عدد است. اگر واژگان None نباشد این پارامتر نادیده گرفته می‌شود.

اگر مراحل بالا را از تغییر min_df به 0.001 تکرار کنیم، سطح آستانه کاهش می‌یابد و کلمات بیشتری به جای 261 کلمه قبلی به ویژگی‌ها اضافه می‌شوند، یعنی 704 کلمه.

```
[6]: #Lets find vocabulary/features
features = vect.get_feature_names()
features

[6]: ['01',
 '10',
 '11',
 '12',
 '13',
 '14',
 '15',
 '16',
 '39',
 'adventures',
 'alex',
 'alex cross',
 'america',
 'american',
 'angel',
 'angels'
```

...

تشابه کسینوسی بین عناوین:

کارهایی که باید انجام دهیم:

- مقداردهی cosine similarity به ماتریس عنوان
- استخراج ویژگی‌ها از عنوان کتاب
- استفاده از شباهت کسینوس بین این عنوان و سایر عناوین تا عنوان 10 کتاب برتر توصیه می‌شود.

[7]:
```
from sklearn.metrics.pairwise import cosine_similarity
cosine_sim_titles = cosine_similarity(title_matrix, title_matrix)
cosine_sim_titles.shape
```

[7]:
```
(10000, 10000)
```

[8]:
```
#Get books which are similar to a given title
title_id = 100
books_title['title'].iloc[title_id]
```

[8]:
```
'Me Talk Pretty One Day'
```

[9]:
```
#Find out what features have been considered  by the vectorizer for a given title ?
feature_array = np.squeeze(title_matrix[title_id].toarray()) #squeeze activity matrix into array
idx = np.where(feature_array > 0)
idx[0]
[features[x] for x in idx[0]]
```

[9]:
```
['day', 'pretty']
```

[10]:
```
# Find index of feature
idx[0]
```

[10]:
```
array([ 63, 179], dtype=int64)
```

[11]:
```
#Cosine similarity with other similar titles
n = 15 #how many books to be recommended
top_n_idx = np.flip(np.argsort(cosine_sim_titles[title_id,]), axis = 0)[0:n]
top_n_sim_values = cosine_sim_titles[title_id, top_n_idx]
top_n_sim_values
```

[11]:
```
array([1.        , 0.70710678, 0.70710678, 0.70710678, 0.70710678,
       0.70710678, 0.70710678, 0.70710678, 0.70710678, 0.70710678,
       0.70710678, 0.70710678, 0.70710678, 0.70710678, 0.70710678])
```

[12]:
```
#find top n with values > 0
top_n_idx = top_n_idx[top_n_sim_values > 0]
#Matching books
books_title['title'].iloc[top_n_idx]
```

[12]:
```
100                                Me Talk Pretty One Day
3729                                           Labor Day
988                                  The Day of the Jackal
836                               Every Day (Every Day, #1)
2348    No Easy Day: The Firsthand Account of the Miss...
3311                                           Pretty Baby
6804                     Graduation Day (The Testing, #3)
6886                                Day Watch (Watch #2)
5765                           The Given Day (Coughlin #1)
783                                      For One More Day
9210            Beyond Exile (Day by Day Armageddon,# 2)
9703    The Pretty Committee Strikes Back (The Clique,...
9637                                  Day 21 (The 100, #2)
7330                                       Pretty in Plaid
7707                             A Grown-Up Kind of Pretty
Name: title, dtype: object
```

[13]:
```
# lets wrap the above code in a function
def return_sim_books(title_id, title_matrix, vectorizer, top_n = 10):

    # generate sim matrix
    sim_matrix = cosine_similarity(title_matrix, title_matrix)
    features = vectorizer.get_feature_names()

    top_n_idx = np.flip(np.argsort(sim_matrix[title_id,]),axis=0)[0:top_n]
    top_n_sim_values = sim_matrix[title_id, top_n_idx]

    # find top n with values > 0
    top_n_idx = top_n_idx[top_n_sim_values > 0]
    scores = top_n_sim_values[top_n_sim_values > 0]


    # find features from the vectorized matrix
    sim_books_idx = books_title['title'].iloc[top_n_idx].index
    words = []
    for book_idx in sim_books_idx:
        try:
            feature_array = np.squeeze(title_matrix[book_idx,].toarray())
        except:
            feature_array = np.squeeze(title_matrix[book_idx,])
        idx = np.where(feature_array > 0)
        words.append([" , ".join([features[i] for i in idx[0]])])

    # collate results
    res = pd.DataFrame({"book_title" : books_title['title'].iloc[title_id],
           "sim_books": books_title['title'].iloc[top_n_idx].values,"words":words,
           "scores":scores}, columns = ["book_title","sim_books","scores","words"])


    return res
```

[14]:
```
vect = CountVectorizer(analyzer='word',ngram_range=(1,2),stop_words='english', min_df = 0.001)
vect.fit(books_title['title'])
title_matrix = vect.transform(books_title['title'])
print(books_title['title'][10])
return_sim_books(10,title_matrix,vect,top_n=10)
```

```
The Kite Runner
```

[14]:

	book_title	sim_books	scores	words

با استفاده از CountVectorizer می‌توانیم متن را از عناوین کتاب استخراج کرده و عناوین مشابه را پیشنهاد دهیم. محدودیتی که با آن روبرو شدم برای عناوینی مانند "kite" و "dinner" بود. به دلیل وجود کلمات کمیاب مانند kite و dinner در مجموعه داده، امکان توصیه کتاب‌های مشابه وجود ندارد، بنابراین ما باید به دنبال روش‌های بهتر برای توصیه‌ها باشیم.

در مراحل بعدی، از Tf-Idf و Rake استفاده خواهم کرد تا ببینم کدام یک از اینها نتایج بهتری به ما می‌دهد.

استفاده از Tf-Idf Vectorizer:

در اینجا ما وزن کلی یک کلمه در داکیومنت را مفید می‌دانیم، در حالی که با کلمات پرتکرار کار می‌کنیم.

	evil	horizon	of	problem	queen
0	0.517856	0.000000	0.680919	0.517856	0.000000
1	0.605349	0.000000	0.000000	0.000000	0.795961
2	0.000000	0.795961	0.000000	0.605349	0.000000

کارهایی که باید انجام دهیم:

- مقداردهی Tfidf vectorizer و قرار دادن آن در ستون عنوان
- محاسبه تشابه کسینوسی
- همه عناوین را به یک مجموعه مرتبط با شماره فهرست کتاب تبدیل می‌کنیم
- تابعی که توصیه‌های کتاب را براساس نمره تشابه کسینوس عناوین کتاب دریافت می‌کند.

[15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

tf = TfidfVectorizer(analyzer = 'word', ngram_range = (1,2), min_df = 0, stop_words = 'english')
tfidf_matrix = tf.fit_transform(books_title['title'])
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
cosine_sim
```

[15]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

[16]:
```
titles = books_title['title']
indices = pd.Series(books_title.index, index = books_title['title']) #converting all titles into a Series

#Function that gets book recommendations based on the cosine similarity score of book titles
def book_recommendations(title, n):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key = lambda x:x[1], reverse = True)
    sim_scores = sim_scores[1:n+1]
    book_indices = [i[0] for i in sim_scores]
    return titles.iloc[book_indices]
```

[17]:
```
#Recommend n books for a book having index 1
book_index = 10
n = 20

print(books_title['title'][book_index])
book_recommendations(books_title.title[book_index],n)
```

```
The Kite Runner
```

[17]:
```
8946                                        Once a Runner
90                        The Maze Runner (Maze Runner, #1)
375                         The Death Cure (Maze Runner, #3)
945                       The Kill Order (Maze Runner, #0.5)
258                      The Scorch Trials (Maze Runner, #2)
6711    Ultramarathon Man: Confessions of an All-Night...
0                  The Hunger Games (The Hunger Games, #1)
```

توصیه کتاب‌های مشابه بر اساس لیست کتاب‌های خوانده شده:

در نهایت برای استفاده از ریکامندر سیستم می توانیم لیستی از کتاب هایی که خواندیم و تعداد(n) پیشنهاداتی که می خواهیم را به تابع book_recommendaions بدهیم و براساس آن ها به ما n کتاب توصیه میکند.

Input list of books

```
[18]: # you can pass list of book as arguman
      book_recommendations('A Tale of Two Cities',3)
```

```
[18]: 5871    A Tale of Two Cities / Great Expectations
      2697                               Invisible Cities
      1699                    A Tale for the Time Being
      Name: title, dtype: object
```

منابع:

https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd

https://www.kaggle.com/zygmunt/goodbooks-10k (dataset)