Two examples of a content-based recommendation system

Content-based, weighted content-based, Numpy functions

Xue Wang Mar 26 · 4 min read ★

Photo from Michal Matlon on Unsplash

Today I would like to discuss two examples for content-based recommendation systems and some efficient array functions I learn from them. The two examples are

1: Based on item content recommendation

2: Based on weighted content recommendation

I use a simple movie set as an example and would like to focus on the main process and ignore other processes and special cases. Let's get started.

Datasets preparing:

Use the below codes to generate two datasets: movie_df and review_df

```
movie_id=[1,2,3,4,5]
user_id=[100,100,200,300, 400]
movie_title=['toy_story','superman','titannic','follow_me','minari']
```

```
movies={'movie_id':movie_id,'movie_title':movie_title,'genres':genres}
reviews={'movie_id':movie_id,'user_id':user_id,'rating':rating}
movies_df= pd.DataFrame(movies,columns=['movie_id',  'movie_title','gen
reviews_df= pd.DataFrame(reviews,columns=['movie_id', 'user_id', 'ratin
```

The two tables as:

```
movies_df
```

	movie_id	movie_title	genres
0	1	toy story	children,comedy
1	2	superman	drama,children
2	3	titannic	drama,romantic
3	4	follow me	horror
4	5	minari	drama

```
reviews_df
```

	movie_id	user_id	rating
0	1	100	3
1	2	100	2
2	3	200	3
3	4	300	2
4	5	400	2

Method 1: based on the movie content, make a recommendation for each user when the similarity of the content is greater than 0.

recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature.

Every genre is separated by a ",", so we simply have to call the split function on

```
movies_df['genres'] = movies_df.genres.str.split(',')

movies_genres_df = movies_df.copy()
#For every row in the dataframe, iterate through the list of genres and
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        movies_genres_df.at[index, genre] = 1

movies_genres_df.head()
```

	movie_id	movie_title	genres	children	comedy	drama	romantic	horror
0	1	toy story	[children, comedy]	1.0	1.0	0.0	0.0	0.0
1	2	superman	[drama, children]	1.0	0.0	1.0	0.0	0.0
2	3	titannic	[drama, romantic]	0.0	0.0	1.0	1.0	0.0
3	4	follow me	[horror]	0.0	0.0	0.0	0.0	1.0
4	5	minari	[drama]	0.0	0.0	1.0	0.0	0.0

Step 1: Calculate the movie-movie similarity matrix :

```
# Subset using the dummy variables
movie_genres = np.array(movies_genres_df.iloc[:,3:])
# dot product to obtain a movie x movie matrix of similarities
dot_prod_movies_genres = movie_genres .dot(np.transpose(movie_genres ))
```

```
[1., 2., 1., 0., 1.],
[0., 1., 2., 0., 1.],
[0., 0., 0., 1., 0.],
[0., 1., 1., 0., 1.]])
```

The dot product shows the similarity among the movies.

Step 2: Find similar movies: here the standard is that if there is a relationship then choose to recommend as the small dataset.

```
def find_similar_movies(movie_id):
    '''
    INPUT
    movie_id - a movie_id
    OUTPUT
    similar_movies - an array of the most similar movies by title
    '''
    # find the row of each movie id
    movie_idx = np.where(movies_genres_df['movie_id'] == movie_id)[0][



    # find the most similar movie indices - to start I said they need

    similar_idxs = np.where(dot_prod_movies_genres[movie_idx] >=1 )[0]



    # pull the movie titles based on the indices
    similar_movies = np.array(movies_genres_df.iloc[similar_idxs, ]['m
```

which only choose the highest value of the similarity.

```
# only choose the most similar movie
```

np.where() shows the item position (index) in the table.

Make a test for movie_id =1:

```
find_similar_movies(1)
```

```
array(['toy story', 'superman'], dtype=object)
```

A similar movie with toy story is superman, whose genre is children. The result is as expected.

Then get a list of movie name, if the name of the movie is a list:

```
def get_movie_names(movie_ids):
    '''
    INPUT
    movie_ids - a list of movie_ids
    OUTPUT
    movies - a list of movie names associated with the movie_ids
    '''
    movie_lst = list(movies_genres_df[movies_genres_df['movie_id'].isin
    return movie_lst
```

Step 3: Make recommendations for a specific user:

```
def make_recs(user_id):

    recs=np.array([])
    # Pull only the reviews the user has seen
    reviews_temp = reviews_df[reviews_df['user_id'] == user_id]
    movies_temp = np.array(reviews_temp['movie_id'])
```

```
        temp_recs = np.setdiff1d(rec_movies, movie_names)
        recs=list(temp_recs)

```

```
make_recs(100)
```

```
['minari', 'titannic']
```

Both of them are drama, as user 100 has also reviewed drama 'superman', so the recommendation makes sense.

Method 2: Based on weighted content

Method 1 is easy to understand, but it seems the rating information has not been used. Now I want to integrate the info to calculate the weighted genres.

For example, I want to construct weighted genres based on the user's rating. Let's choose user_id=100.

Step 1: Filter out the movies which are rated by user 100 and get the genres only:

```
#Filtering out the movies from the review
movie_id_df=reviews_df.loc[reviews_df['user_id']==user_id]
user_movies = movies_genres_df[movies_genres_df['movie_id'].isin(movie_
user_movies = user_movies.iloc[:,3:]
```

1	1.0	0.0	1.0	0.0	0.0

Step 2: Get weighted genres for this user:

```
rating_df=reviews_df.loc[reviews_df['user_id']==user_id]['rating']
#Dot produt to get weights
userProfile = userMovies.transpose().dot(rating_df)
```

get_weighted_movie_genres.py hosted with ❤ by GitHub view raw

```
children    5.0
comedy      3.0
drama       2.0
romantic    0.0
horror      0.0
dtype: float64
```

Step 3: Get a recommendation (user-reviewed also included)

```
recommendation_array=movie_genres.transpose().dot(userProfile)/(userPro
recommendation_series=pd.Series(recommendation_array).sort_values(ascen
recommendation_keys=[keys for keys, value in recommendation_series.item
rec_movies=movies_df.loc[movies_df['movie_id'].isin(movie_ids)]
```

	movie_id	movie_title	genres
0	1	toy story	[children, comedy]
1	2	superman	[drama, children]
2	3	titannic	[drama, romantic]

the user.

Step 4: Filter the reviewed movies

```
user_reviews = reviews_df[reviews_df['user_id'] == 100]
movies_reviews = np.array(user_reviews['movie_id'])
movie_names = np.array(get_movie_names(movies_reviews))
rec_movies=np.array(rec_movies['movie_title'])
recs=np.setdiff1d(rec_movies, movie_names)
```

```
recs
```

```
array(['titannic'], dtype=object)
```

In this case, as minari's weighted genres for user 100 is 0.2, so it is excluded from the recommendation.

Summary:

- From the two cases, we can see that no reviewed genres will not be recommended, which is the character of a content-based recommendation system. It is highly personalized for the user.
- To some degree, a recommendation system is like an art, and you can also create your criteria to adjust to your target, like in this story I have displayed various criteria even for the same method.
- There are some array functions which I believe are very efficient when dealing with recommendation system, and would like to summarize here again:

and use it.

- 2)**np.dot(np.transpose())**: this dot product can be used not only for the item itself but also for getting the weighted items.
- 3) **np.where()**: here I use it to find the location(index) of the item.
- Of course, you can also extrapolate them to other situations, not only the recommendation system. Because of the limitation, I haven't enough time to show how the functions are efficient, and I explain them in this story in detail.

Thank you for your reading.

Recommendation System Arrays Python Content Based Filtering