인공지능특강(추천시스템) Project 1 Report

2022-26772 장석운

2023.03.29

(Task 1) Genre를 이용한 movie representation

```
# A) movies_w_imgurl.csv에서 영화의 Genre 추출
mv_genre_df = pd.read_csv("./data/movies_w_imgurl.csv")
genres = mv_genre_df["genres"]

num_movie = len(genres) # 총 영화 수
genre_counter = Counter()
for genre in genres:
    glist = genre.split('|')
    genre_counter.update(glist)

# B) Genre의 IDF 계산
#    IDF : numpy.log10(전체 movie 개수 / 각 genre별 movie 개수)
genre_idf = {}
for genre in genre_counter:
    genre_idf[genre] = -1 * np.log10(num_movie / genre_counter[genre])
genre_idf = sorted(genre_idf.items())
```

```
genre_tf_idf = []
for i, genre in enumerate(genres):
    glist = genre.split('|')
    tmp = []
    for ig in genre_idf:
        if ig[0] in glist: # 해당 영화의 장르의 tf-idf 계산
            tmp.append(ig[1])
        else:
            tmp.append(0)
    genre_tf_idf.append(tmp)
```

Movie 별 genre를 split 함수를 이용해 나누고 공식에 따라 genre의 IDF를 계산합니다. Counter() 객체를 이용해 각 genre 별 영화의 개수를 계산합니다. 이 후 영화마다 장르에 따른 TF-IDF를 계산합니다. 영화에 해당하는 장르가 없을 경우 0을 배열에 넣습니다. 결과는 9125개의 영화에 대한 20개 장르 별 TF-IDF가 저장됩니다. (Shape of genre_tf_idf : 9125 x 20)

(Task 2) Tag를 이용한 movie representation 보완

```
# A) tags.csv에서 영화의 tag 추출
mv_tags_df = pd.read_csv("./data/tags.csv")
mvid_tags = mv_tags_df[mv_tags_df.columns[1:3]]
mvid = mv_tags_df["movieId"]
tag = mv_tags_df["tag"]

# 각 tag가 붙은 영화 수, len(tag_counter): 586,
tag_counter = Counter()
for t in tag:
    r = t.split(', ')
    tag_counter.update(r)

# tag가 달린 영화 수 tmovie_len : 689
tmovie_len = len(set(mvid))
print(tmovie_len)

# B) tag 별 IDF 계산
#    IDF : numpy.log10(전체 movie 개수 / 각 tag에 있는 movie count)
tag_idf = Counter()
for t in tag_counter:
    tag_idf[t] = np.log10( tmovie_len / tag_counter[t])
```

```
# C) TF-IDF 계산
#    TF는 한 영화에 대해 tag가 몇 번 등록되었는지에 따라 계산 (한
#    TF공식: n(d, t) / n(d)

# tag가 달린 영화에 어떤 tag가 있는지
movie_tag = {}
for mv in mvid:
    movie_tag[mv] = []

for i in range(len(tag)):
    movie_tag[mvid[i]] = movie_tag[mvid[i]] + tag[i].split(', ')
```

```
for mv in mv_list:
    tmp = []
    if mv in movie_tag:
        for tag in tag_counter:
            tf = len([t for t in movie_tag[mv] if t == tag]) / len(movie_tag[mv]) # TF = (n(d, t) / n(d))
            if tag in movie_tag[mv]:
                tmp.append(tag_idf[tag] * tf)
            else:
                tmp.append(0)
    else:
        tmp = no_tag_li
    tag_tf_idf.append(tmp)
```

```
# 최종 movie representation
tf_idf = np.concatenate((np.array(genre_tf_idf), np.array(tag_tf_idf)), axis=1)
print(tf_idf.shape)
```

```
(9125, 606)
```

Counter()와 set()을 이용하여 각 tag에 있는 영화의 개수와 tag가 담긴 영화의 개수를 구합니다. 이를 이용해 IDF를 구한 후 영화마다 tag에 따른 TF-IDF를 계산합니다. Unique한 tag의 개수는 586개이므로 tag에 따른 TF-IDF 리스트의 크기는 9125 x 586이 됩니다. 여기에 앞서 구한 genre에 따른 TF-IDF 를 concatenate 시키면 최종적으로 (9125 x 606) 사이즈의 movie representation TF-IDF matrix가 생성됩니다.

(Task 3) 두 movie 사이의 cosine similarity 계산

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
mv_cossim = cosine_similarity(tf_idf, tf_idf)
```

Scikit-learn 라이브러리의 cosine_similarity() 함수를 이용하여 movie 사이의 cosine similarity를 계산합니다. (9125 x 9125) 사이즈의 numpy matrix가 만들어집니다.

(Task 4) Content-based 추천 점수 계산

```
ratings = pd.read_csv("./data/ratings.csv")
ratings.head()
user_rate = {}
for i in range(len(ratings["userId"])):
    user = ratings["userId"][i]
    mvid = ratings["movieId"][i]
    user_rating = ratings["rating"][i]
    if user in user_rate:
        user_rate[user].append([ mvid, user_rating])
    else:
        user_rate[user] = [ [ mvid, user_rating] ]
```

```
# mv_cossim로부터 user_sim 추출
user_sim = {}
# user_sim_sum = {}
for user in user_rate:
    mvids = np.array(user_rate[user][0])
    sim = []
    for mvid in mvids:
        sim.append(mv_cossim[mv_mvid_map[mvid]]) # 1 x 9125
    user_sim[user] = np.array(sim)
    # user_sim_sum[user] = np.sum(np.array(sim), axis = 0)
```

```
#user id의 전체 영화에 대한 추정 점수 계산
user_score = []
for user in user_rate:
    score = np.matmul(user_sim[user].T, user_rate[user][1]) / (np.sum(user_sim[user], axis = 0) + 1) # 전체 영화 추정 점수 계산
    user_score.append(score)

print("total user score shape : ", user_score[1].shape) # user의 movie별 score
print("total users : ", len(user_score))
```

```
total user score shape :  (9125,)
total users :  671
```

user마다 rate를 준 movie와 그 rate를 user_rate 딕셔너리에 저장, 또 user마다 rate를 준 movie의 user_sim을 앞서 계산한 mv_cossim에서 가져와 user_sim 딕셔너리에 저장한다. 이 후 user별로 전체 영화에 대한 추정 점수를 공식대로 계산한다. 결과는 user_score 리스트에 저장되고 그 크기는 user의 수 671 x 영화 수 9125가 된다.

(어려웠던 점 및 해결 방법)

주어진 TF-IDF 공식의 의미를 파악하는 것이 어려웠습니다. Genre에서 계산한 TF-IDF가 무엇인지, tag별 TF-IDF가 무엇을 의미하는지 수업시간에 준 자료를 보며 파악하였습니다. 또한 project pdf 자료에 나온 단계별 matrix shape와 그림 예시를 보면서 구현이 맞는지 검증할 수 있었습니다.

(느낀 점)

기본적인 content-based 알고리즘을 구현해보면서 수업 시간에 배운 content-based 기법에 대해 알 수 있었다. TF-IDF의 의미가 무엇인지 파악하면서 구현을 해 나감으로써 단순 본 과제를 구현만 하고 끝나는 것이 아닌 다른 분야에도 적용할 수 있을 것이라고 생각한다.