메타데이터 개수 증가를 이용한 콘텐츠 기반 영화 추천 시스템의 정확도 향상 테스트

최다정[O], 서진경[*], 백주련(교신저자)[*]
[O]평택대학교 데이터정보학과,
[*]평택대학교 데이터정보학과
e-mail: dajung2020@ptu.ac.kr[O], sjk2700@naver.com[*], jrpaik@ptu.ac.kr[*]

Accuracy Improvement Test for Contents-based Movie Recommendation System by Increasing Metadata

Da-jeong Choi[O], Jin-kyeong Seo[*], Juryon Paik(Corresponding Author)[*]
[O]Dept. of Digital Information & Statistics, Pyeongtaek University,
[*]Dept. of Digital Information & Statistics, Pyeongtaek University

● 요 약 ●

콘텐츠 기반 추천 시스템은 대표적인 추천 모델 방법 중 하나이다. 하지만 콘텐츠 기반 추천 시스템은 사용자 관련 메타데이터를 고려하기보다 내용 관련 메타데이터에만 의존하는 경향이 있다. 본 논문에서는 영화의 특징을 담고 있는 메타데이터를 이용해 추천 시스템을 간단히 구현하고, 추천한 영화와 사용자의 영화 평점을 이용해 추천 시스템의 정확도를 측정하였다. 영화 메타데이터 keywords, genres, cast의 개수를 늘려가며 정확도가 변화하는지 알아보았다. 메타데이터 각각의 개수가 증가하면 정확도도 향상할 것이라고 기대했으나 큰 차이가 나타나지 않았다. 모델 평가 결과, 미세한 차이지만 영화 메타데이터를 상위 3개씩 추출해 영화를 추천했을 때의 정확도가 1.2100318041248186으로 가장 높았다.

키워드: 메타데이터(metadata), 콘텐츠 기반 추천 시스템(contents-based recommender system)

I. Introduction

추천 시스템(recommender system)이란 과거 데이터를 바탕으로 사용자에게 필요한 정보나 제품을 추천하여 제시해 주는 시스템이다. 대표적으로 넷플릭스, 유튜브, 아마존 등 OTT 서비스(over-the-top media service)와 온라인 쇼핑몰에서 사용되고 있다. 추천 시스템의 주요 알고리즘에는 협업 필터링(collaborative filtering), 콘텐츠 기반 필터링(content-based filtering), 지식 기반 필터링(knowledge-based filtering), 딥러닝(deep learning) 추천 기술, 하이브리드(hybrid) 기술이 있다.

본 논문에서는 콘텐츠 기반 추천 시스템을 적용하여 영화 메타데이터로 영화를 추천하고, 영화 메타데이터의 개수가 증가함에 따라 정확도가 향상하는지 사용자의 영화 평점을 이용해 테스트해보고자 한다.

II. The Proposed Scheme

1. 데이터

1.1 데이터 수집

본 논문에서 영화 추천을 위해 사용한 데이터는 Kaggle에 공개된 The Movies Dataset이다. The Movies Dataset 중 movies_metadata, credits, keywords를 사용하였다.

그리고 사용자의 영화 평점 데이터를 추가하기 위해 GroupLens Research에서 제공한 MovieLens 100K Dataset을 사용하였다. MovieLens 100K Dataset 중 u.item, u.data를 사용하였다. MovieLens 데이터는 미네소타 대학교의 GroupLens Research Project에서 MovieLens 웹사이트를 통해 1997년 9월 19일부터 1998년 4월 22일까지 7개월간 수집되었다. 사용자 평가가 20개 미만이거나 완전한 인구 통계가 없는 데이터는 이 데이터 세트에서 정보가 제거되었다.

1.2 데이터 탐색

movies_metadata는 영화 제목, 장르, 개요, 개봉날짜, 상영 시간, 예산, 수입, 제작사 등 많은 메타데이터를 제공하는 데이터셋이다. 본 논문에서 영화를 추천하기 위해 사용하는 메타데이터보다 필요 없는 메타데이터가 더 많으므로 총 24개의 메타데이터 중 8개만 남겨놓았다. credits 데이터셋는 영화에 출연한 배우와 영화에 참여한 제작진들의 아이디, 맡은 역할, 성별, 이름, 섭외 순위 등을 제공한다. keywords 데이터셋는 영화의 키워드들을 제공한다. 데이터를 정제하지 않은 The Movies Dataset의 데이터타입은 Table 1, 2, 3과 같다.

Table 1. movies_metadata의 데이터&데이터타입

Item	Value
id	영화의 아이디, object
title	영화의 제목, object
genres	영화의 장르, object
overview	영화의 개요, object
release_date	영화의 개봉날짜, object
runtime	영화의 상영 시간, float64
vote_average	영화의 평점 평균, float64
vote_count	영화의 평점 투표수, float64

Table 2. credits의 데이터&데이터타입

Item	Value
cast	영화의 출연진 정보, object
crew	영화의 제작진 정보, object
id	영화의 아이디, int64

Table 3. keywords의 데이터&데이터타입

Item	Value
id	영화의 아이디, int64
keywords	영화의 키워드, object

movies_metadata, credits, keywords의 id 데이터타입이 movies_metadata만 다르므로 같은 int 타입으로 변경, 결측값을 제거하고 저장한다.

u.item 데이터셋는 1,682개의 영화의 아이디와 제목을 제공한다. u.data는 1,682개의 영화에 대해 943명의 사용자로부터 받은 100,000개의 평가 점수를 제공한다. 평가는 1점부터 5점까지의 척도로 부여되어 있다. 각 사용자는 최소 20편의 영화를 평가하였다. 데이터를 정제하지 않은 MovieLens 100K Dataset의 데이터타입은 Table 4, 5와 같다.

Table 4. u.item의 데이터&데이터타입

Item	Value
movie_id	영화의 아이디, int64
title	영화의 제목, object

Table 5. u.data의 데이터&데이터타입

Item	Value
user_id	사용자 아이디, int64
movie_id	영화의 아이디, int64
rating	사용자가 부여한 영화의 평점, int64

1.3 데이터 전처리

데이터 전처리를 하기 전, 필요한 데이터를 한곳에 모으기 위해 movies_metadata, credits, keywords를 id를 기준으로 하나의 데이터프레임으로 병합하였다. 병합한 데이터프레임을 df라고 하겠다.

Fig. 1의 내용은 다음과 같다. 메타데이터 cast, crew, keywords, genres는 문자열화 된 리스트 타입이다. 따라서 literal_eval() 함수를 사용해 파이썬에서 제공하는 기본 데이터 타입인 리스트로 변환하였다. 감독, 배우, 키워드, 장르 메타데이터를 사용하여 영화를 추천하기 때문에 crew에서 감독만 추출해 director에 저장하였다. 그리고 keywords, genres, cast는 리스트의 원소인 사전형의 key가 name인 데이터에 대한 상위 3개를 추출하는 방법이 같으므로 함수를 정의해서 추출하였다.

유사도를 계산하기 전 문서 벡터화를 위해 cast, director, keywords, genre 4개의 특징을 가진 메타데이터를 하나의 텍스트처럼 합쳐서 soup에 저장해주었다.

```
features = ['cast','crew','keywords','genres']

for feature in features:
    df[feature] = df[feature].apply(literal_eval)

def get_director(x):
    for crew in x:
        if (crew['job']=='Director') | (crew['job']=='director'):
            return crew['name']
        return np.nan

df['director'] = df['crew'].apply(get_director)

def get_top_three(x):
    if isinstance(x, list):
        names = [element['name'] for element in x]
        if len(names) > 3:
            names = names[:3]
        return names
    return []

df['keywords'] = df['keywords'].apply(get_top_three)
df['cast'] = df['cast'].apply(get_top_three)
df['genres'] = df['genres'].apply(get_top_three)

def sanitize(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ""

features = ['cast','director','keywords','genres']

for feature in features:
    df[feature] = df[feature].apply(sanitize)

def create_soup(x):
    text = ''
    text = ' '.join(x['keywords']) + ' '
    text = text + ' '.join(x['cast']) + ' '
    text = text + x['director'] + ' '
    text = text + ' '.join(x['genres'])
    return text

df['soup'] = df.apply(create_soup, axis=1)
df = df[:20000]
```

Fig. 1. 데이터 전처리(1)

Fig. 2의 내용은 다음과 같다. u.data에 있는 사용자의 영화 평점 데이터를 가져오기 위해 u.item의 title을 전처리해 title을 기준으로 df와 병합하였다. movies_metadata의 title에는 영화의 개봉연도가 포함되어있지 않지만, u.item의 title에는 개봉연도가 적혀있기 때문에 개봉연도를 제외한 제목만을 포함한 title로 변경해 병합시켜주었다. 병합한 후에 중복된 타이틀을 제거하였다.

```
mt = []
for i in range(len(movies)):
    mt.append(movies.iloc[i]['title'][:-7])

movies['Title'] = mt
movies = movies.drop('title', axis=1)
m_cols = ['movie_id', 'title']
movies.columns = m_cols

df = df.merge(movies, on='title')
df = df.drop_duplicates('title')

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

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df['soup'])
count_sim = cosine_similarity(count_matrix, count_matrix)

df = df.reset_index()

indices = pd.Series(df.index, index=df['title']).drop_duplicates()

df2 = df.copy()
df2 = pd.merge(df2, ratings)

def content_recommender(user_id, count_sim, dataset, indices):
    m_list = dataset[dataset['user_id']==user_id]
    r_max = m_list['rating'].max()
    title = m_list[m_list['rating']==r_max].iloc[0]['title']
    idx = indices[title]
    sim_scores = count_sim[idx]
    sim_scores = enumerate(sim_scores)
    sim_scores = list(sim_scores)
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]
    movies_indices = [i[0] for i in sim_scores]
    return df['title'].iloc[movies_indices]

print("Input Your id")
user_id = int(input())

p_list = content_recommender(user_id, count_sim, df2, indices)

from sklearn.model_selection import train_test_split

X = df2[['index','title','movie_id','user_id','rating']].copy()
y = df2['user_id']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=1004,
                                                    stratify=y)
```

Fig. 2. 데이터 전처리(2)

문서 벡터화를 위해 4개의 특징을 하나의 텍스트처럼 합친 soup 객체의 데이터를 sklearn의 CountVectorizer()와 fit_transform()을 사용해 벡터를 생성하였다. 입력된 user_id가 가장 높게 평가한 영화와 배우, 감독, 키워드, 장르가 비슷한 영화를 추천하기 위해 모든 문서에 자주 등장하는 단어에 페널티를 주는 TfidfVectorizer가 아닌 CountVectorizer를 사용해 자주 등장하는 단어에 가중치를 주게 한 다음, 유사도를 계산하였다. 그리고 df와 u.data를 movie_id를 기준으로 병합하여 df2라는 데이터프레임으로 저장하였다. 사용자의 아이디를 입력받고 사용자가 평가한 영화 중 가장 높게 평가한 영화를 뽑고, 뽑힌 영화와 유사도가 높은 상위 10개의 영화를 추천하였다.

모델을 학습하고 이를 평가하기 위해서 train, test 데이터를 구분하였다. df2에서 index, title, movie_id, user_id, rating 데이터를 복사하고 user_id를 기준으로 삼았다. 층화추출법을 사용하고 train 데이터 75%, test 데이터 25%로 나누었다.

2. 모델 평가

2.1 모델 평가

```
def RMSE(y_true, y_pred):
    result = np.sqrt(np.mean((np.array(y_true)-np.array(y_pred))**2))
    return result

def evaluation(model):
    id_pairs = zip(X_test['user_id'], X_test['movie_id'])
    y_pred = np.array([model(user, movie) for (user, movie) in id_pairs])
    y_true = np.array(X_test['rating'])
    return RMSE(y_true, y_pred)

train_mean = X_train.groupby(['movie_id'])['rating'].mean()
train_mean = pd.DataFrame(train_mean)
train_mean.reset_index()
train_mean = pd.merge(train_mean, X_train[['title', 'movie_id']],
                      on='movie_id')

def level0Model(user_id, movie_id):
    rating = 0
    res = content_recommender(user_id, count_sim, X_train, indices)
    res = list(res)
    for j in range(10):
        try:
            rating +=train_mean[train_mean['title']==res[j]].iloc[0]['rating']
        except:
            rating += 3.0
    return rating/10

evaluatedValue = evaluation(level0Model)
```

Fig. 3. 모델 평가

```
In [7]: evaluatedValue
Out[7]: 1.2100318041248186
```

Fig. 4. 모델 평가 결과

Fig. 3의 내용은 다음과 같다. 추천된 10개의 영화 데이터를 가지고 평점을 예측하였다. 평가는 0에 가까울수록 정확도가 높은 RMSE(평균 제곱근 오차)로 하였다. 이용한 메타데이터 cast, keywords, genre의 개수를 상위 3개씩 추출하여 평가한 RMSE의 결과는 1.2100318041248186이다. 하지만 cast, keywords, genres를 3개로 설정하고 추천하는 것만으로는 정확한지 알 수 없다.

2.2 데이터 개수 조절을 통한 평가

```
In [12]: evaluatedValue
Out[12]: 1.2161355174913229
```

Fig. 5. 메타데이터 상위 4개를 이용한 모델 평가 결과

```
In [14]: evaluatedValue
Out[14]: 1.214177062594141
```

Fig. 6. 메타데이터 상위 5개를 이용한 모델 평가 결과

```
In [16]: evaluatedValue
Out[16]: 1.228856775041277
```

Fig. 7. 메타데이터 상위 6개를 이용한 모델 평가 결과

cast, keywords, genres의 개수를 상위 4개씩으로 조절했을 때의 RMSE는 Fig. 5에 있는 1.2161355174913229이다. 상위 5개, 6개로

늘린 경우에도 Fig. 6, Fig. 7같이 비슷하거나 더 높은 결과를 보였다. 미세한 차이지만 cast, keywords, genres의 개수를 3개로 설정한 후 추천하는 방법의 정확도가 가장 높다는 것을 알 수 있다.

III. Conclusions

영화 메타데이터를 이용하여 영화를 추천해보았다. 가중치를 줄 메타데이터 각각의 개수를 증가시키며 영화를 추천하는 방식을 사용하면 정확도가 높아지기를 기대했으나, 큰 차이를 보지 못하였다. 차후 연구에서는 영화 추천의 정확도에 영향을 미치는 데이터를 더 추가하고 다른 모델들과 비교하여 모델의 정확도를 더 높이는 방향으로 나아가고자 한다.

ACKNOWLEDGEMENT

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2021R1F1A1064073).

REFERENCES

[1] Lim, Il. (2020). Personalized recommendation system using Python.

[2] BANIK, R. (2018). HANDS-ON RECOMMENDATION SYSTEMS WITH PYTHON: Start building powerful and personalized, ... recommendation engines with python. Place of publication not identified: PACKT Publishing Limited.

[3] Kaggle, https://www.kaggle.com/

[4] GroupLens, https://grouplens.org/

[5] Visit Korea Byul, Lee Hye-woo, and Lee Ji-hyung. (2015). Content-based TV program recommendation system using age and program plot. Proceedings of the Korean Society for Computer Information and Information Science, 23(1), 51-54.