# 영화 추천 시스템

인공지능

**2017107018** 정승원

# 목차

1. 인구 통계 필터링

2. 콘텐츠 기반 필터링

3. 협업 필터링

# Part 1

## 인구 통계 필터링

# 시작하기 전에 필요한 것

영화에 점수를 매기거나 등급을 매기기 위한 측정 기준이 필요합니다.

모든 영화의 점수를  계산해야 합니다.

점수를 정렬하고 사용자에게 가장 좋은 평가를 받은 동영상을 추천해야 합니다.


투표수 : 3


투표수 : 40

$$Weighted\ Rating\ (WR) = (\frac{v}{v+m}, R) + (\frac{m}{v+m}, C)$$

001

```
In [4]: C= df2['vote_average'].mean()
        C
Out[4]: 6.092171559442011
```

002

```
In [5]: m= df2['vote_count'].quantile(0.9)
        m
Out[5]: 1838.4000000000015
```

003

```
In [6]: q_movies = df2.copy().loc[df2['vote_count'] >= m]
        q_movies.shape
Out[6]: (481, 23)
```

In [7]:

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [8]:

```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

In [9]:

```
#Sort movies based on score calculated above
q_movies = q_movies.sort_values('score', ascending=False)

#Print the top 15 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

| | title | vote_count | vote_average | score |
|---|---|---|---|---|
| 1881 | The Shawshank Redemption | 8205 | 8.5 | 8.059258 |
| 662 | Fight Club | 9413 | 8.3 | 7.939256 |
| 65 | The Dark Knight | 12002 | 8.2 | 7.920020 |
| 3232 | Pulp Fiction | 8428 | 8.3 | 7.904645 |
| 96 | Inception | 13752 | 8.1 | 7.863239 |
| 3337 | The Godfather | 5893 | 8.4 | 7.851236 |
| 95 | Interstellar | 10867 | 8.1 | 7.809479 |
| 809 | Forrest Gump | 7927 | 8.2 | 7.803188 |
| 329 | The Lord of the Rings: The Return of the King | 8064 | 8.1 | 7.727243 |
| 1990 | The Empire Strikes Back | 5879 | 8.2 | 7.697884 |

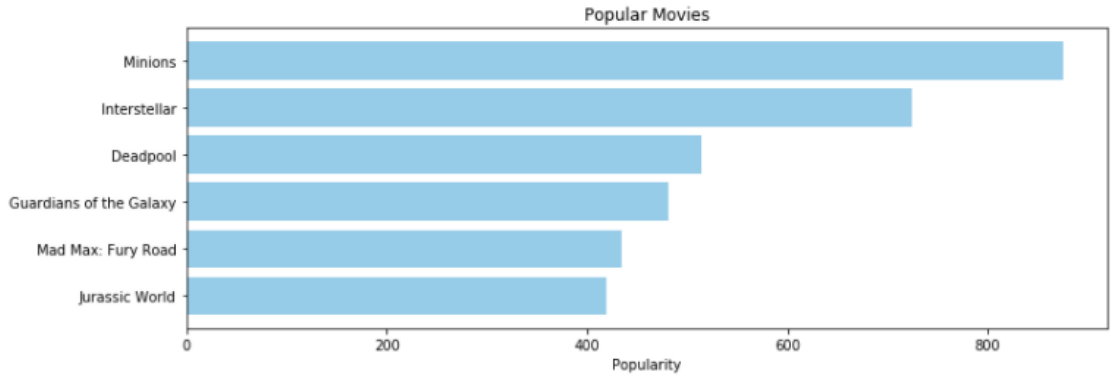
In [10]:

```
pop= df2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

Out[10]:

```
Text(0.5,1,'Popular Movies')
```



# Part 2

## 콘텐츠 기반 필터링

# 줄거리 설명 기반 추천

In [11]:

```
df2['overview'].head(5)
```

Out[11]:

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

In [12]:

```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
df2['overview'] = df2['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

Out[12]:

```
(4803, 20978)
```

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

In [13]:

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [14]:

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

In [15]:

```python
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

In [16]:

```python
get_recommendations('The Dark Knight Rises')
```

Out[16]:

```
65                          The Dark Knight
299                          Batman Forever
428                          Batman Returns
1359                                 Batman
3854    Batman: The Dark Knight Returns, Part 2
119                          Batman Begins
2507                             Slow Burn
9           Batman v Superman: Dawn of Justice
1181                                   JFK
210                          Batman & Robin
Name: title, dtype: object
```

# 장르 및 키워드 기반 추천

In [19]:

```
# Get the director's name from the crew feature. If director is not listed, return N
aN
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

In [20]:

```
# Returns the list top 3 elements or entire list; whichever is more.
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If n
o, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

In [25]:

```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + '
'.join(x['genres'])
df2['soup'] = df2.apply(create_soup, axis=1)
```

In [26]:

```
# Import CountVectorizer and create the count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df2['soup'])
```

In [27]:

```
# Compute the Cosine Similarity matrix based on the count_matrix
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

In [28]:

```
# Reset index of our main DataFrame and construct reverse mapping as before
df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])
```

In [29]:

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

Out[29]:

```
65              The Dark Knight
119               Batman Begins
4638    Amidst the Devil's Wings
1196               The Prestige
3073           Romeo Is Bleeding
3326              Black November
1503                      Takers
1986                      Faster
303                     Catwoman
747               Gangster Squad
Name: title, dtype: object
```

# Part 3

협업 필터링

# 아이템 기반 협업 필터링

| | The Avengers | Sherlock | Transformers | Matrix | Titanic | Me Before You |
|---|---|---|---|---|---|---|
| A | 2 | | 2 | 4 | 5 | 2.94* |
| B | 5 | | 4 | | | 1 |
| C | | | 5 | | 2 | 2.48* |
| D | | 1 | | 5 | | 4 |
| E | | | 4 | | | 2 |
| F | 4 | 5 | | 1 | | 1.12* |
| Similarity | -1 | -1 | 0.86 | 1 | 1 | |

# 잠재요소

**사용자나 아이템이 가지고 있는 속성이나 개념을 설명하는 아이디어**

In [31]:

```
from surprise import Reader, Dataset, SVD, evaluate
reader = Reader()
ratings = pd.read_csv('../input/the-movies-dataset/ratings_small.csv')
ratings.head()
```

Out[31]:

| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 31 | 2.5 | 1260759144 |
| 1 | 1 | 1029 | 3.0 | 1260759179 |
| 2 | 1 | 1061 | 3.0 | 1260759182 |
| 3 | 1 | 1129 | 2.0 | 1260759185 |
| 4 | 1 | 1172 | 4.0 | 1260759205 |

In [34]:

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Out[34]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f87050c8748>
```

In [35]:
```
ratings[ratings['userId'] == 1]
```

Out[35]:

| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 31 | 2.5 | 1260759144 |
| 1 | 1 | 1029 | 3.0 | 1260759179 |
| 2 | 1 | 1061 | 3.0 | 1260759182 |
| 3 | 1 | 1129 | 2.0 | 1260759185 |
| 4 | 1 | 1172 | 4.0 | 1260759205 |
| 5 | 1 | 1263 | 2.0 | 1260759151 |
| 6 | 1 | 1287 | 2.0 | 1260759187 |
| 7 | 1 | 1293 | 2.0 | 1260759148 |
| 8 | 1 | 1339 | 3.5 | 1260759125 |
| 9 | 1 | 1343 | 2.0 | 1260759131 |
| 10 | 1 | 1371 | 2.5 | 1260759135 |
| 11 | 1 | 1405 | 1.0 | 1260759203 |
| 12 | 1 | 1953 | 4.0 | 1260759191 |
| 13 | 1 | 2105 | 4.0 | 1260759139 |
| 14 | 1 | 2150 | 3.0 | 1260759194 |
| 15 | 1 | 2193 | 2.0 | 1260759198 |
| 16 | 1 | 2294 | 2.0 | 1260759108 |
| 17 | 1 | 2455 | 2.5 | 1260759113 |
| 18 | 1 | 2968 | 1.0 | 1260759200 |
| 19 | 1 | 3671 | 3.0 | 1260759117 |

In [36]:
```
svd.predict(1, 302, 3)
```

Out[36]:
```
Prediction(uid=1, iid=302, r_ui=3, est=2.756291780279027, details={'was_impossible': False})
```

# 결론

인구 통계 필터링은 모든 사용자에게 일반화된 추천을 제공하지만,
특정 사용자의 관심사와 취향에는 민감하지 않다.

콘텐츠 기반 필터링은 특정 아이템을 기준으로 유사한 아이템을 추천한다. 하지만
누구나 같은 추천을 받을 수 있기 때문에 사용자의 개인적인 취향을 포착하지 못한다.

협업 필터링은 사용자가 평가한 아이템과의 유사성을 기준으로 아이템을 추천한다.

# 원본 캐글 링크

**https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system**

# 감사합니다.