Netflix
Movie Recommendation

발 표 자 : 안 혜 림

Content

Introduction

추천 시스템 소개

Recommendation models

- 협업 필터링을 통한 추천
- 피어슨 R 상관관계 통한 추천

Data manipulation

- Data loading
- Data viewing
- Data cleaning
- Data slicing
- Data mapping

01

추천 시스템
소개

Introduction

자신과 타인의 행동을 기반으로, 데이터에서 배워 최고의 TV 프로그램을 사용자에게 추천
넷플릭스: 특정 고객에게 영화 추천 서비스 제공 -> 고객 이탈률 4% 이하

추천 알고리즘

이번 분석에서는 CF 와 상관분석을 활용

-구매/소비 패턴이 비슷한 사용자를 한 집단으로 보고 그 집단에 속한 소비자들의 취향을 추천하는 방식

- UBCF(User Based CF): 패턴이 비슷한 사용자를 기반으로 상품 추천

- IBCF(Item Based CF): 상품을 기반으로 연관성이 있는 상품 추천

Collaborative Filtering : CF
협업 필터링

Knowledge-Based Filtering: KB
지식기반 필터링

- 특정 분야에 대한 전문가의 도움을 받아서 그 분야에 대한 전체적인 지식 구조를 만들고 이를 활용하는 방식

- 뉴스, 책 등 텍스트의 내용을 분석해서 추천하는 방식

- 소비자가 소비하는 제품 중 텍스트 정보가 많은 제품을 대상으로 함

- 텍스트 중에서 형태소(명사,동사 등)를 분석하여 핵심 키워드를 분석하는 기술이 핵심

Content-Based Filtering: CB
내용기반 필터링

협업 필터링(CF)의 유사도 계산 방법

- 상관계수 유사도: 피어슨 상관계수 이용

- 코사인 유사도: 두 벡터 사이의 각도

- Jaccard 유사도: 이진화 자료(binary data) 대상 유사도 계산

- 유클리드 거리 계산법: 거리 기반 유사도 계산

02

데이터 분석

Data manipulation

넷플릭스로부터 직접 받은 이 데이터셋은 총 4개의 텍스트 데이터로 구성되어 있음. 각 파일은 20M 열, 4K 영화, 400K 고객들을 포함하며 전체 데이터를 총 합치면 총 17K영화와 500K 이상의 고객 정보를 포함하고 있음.

Data loading

```
import pandas as pd
import numpy as np
import math
import re
from scipy.sparse import csr_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
sns.set_style("darkgrid")
```

각 데이터 파일은 아래의 column을 포함:

- Movie ID
- Customer ID
- Rating (1 to 5)
- 그들이 Rating을 얻은 날짜

다른 파일은 이름, 개봉날짜 같은 영화 배경에 대한 ID 매핑(하나의 값을 다른 값으로 대응)이 포함됨

① Surprise 패키지 : 추천시스템 패키지
이 패키지를 활용하면 보다 쉽게 API를 활용하여 추천 시스템을 구축할 수 있음

Data loading

```
# Skip date
df1 = pd.read_csv('../input/combined_data_1.txt', header = Non
e, names = ['Cust_Id', 'Rating'], usecols = [0,1])

df1['Rating'] = df1['Rating'].astype(float)

print('Dataset 1 shape: {}'.format(df1.shape))
print('-Dataset examples-')
print(df1.iloc[::5000000, :])
```

```
Dataset 1 shape: (24058263, 2)
-Dataset examples-
          Cust_Id  Rating
0               1:     NaN
5000000   2560324     4.0
10000000  2271935     2.0
15000000  1921803     2.0
20000000  1933327     3.0
```

첫 번째 데이터 파일 불러오기

① header(헤더) : 데이터 파일에서 여러가지 값들이 어떤 의미를 갖는지 표시한 행으로, header=None이므로 데이터 속성부터 바로 가져오겠다는 의미

② astype(float) : '실수 ' 형태로 저장하겠다는 의미

③ Cust_ID(고객 아이디)와 Rating(평점) 열을 나타냄

Data loading

```
# load less data for speed

df = df1
#df = df1.append(df2)
#df = df.append(df3)
#df = df.append(df4)

df.index = np.arange(0,len(df))
print('Full dataset shape: {}'.format(df.shape))
print('-Dataset examples-')
print(df.iloc[::5000000, :])
```

① ②

```
Full dataset shape: (24058263, 2)
-Dataset examples-
          Cust_Id  Rating
0               1:     NaN
5000000   2560324     4.0
10000000  2271935     2.0
15000000  1921803     2.0
20000000  1933327     3.0
```

2-4 데이터도 1과 동일하게 불러온 후
속도를 위해 적은 데이터만 로드함

① append() : 덧붙인다는 뜻으로 괄호()안에 값을 입력하면 새로운 요소를 array 맨 끝에 객체로 추가함
(현재는 #으로 주석처리한 상태)

② np.arrange(0, len(df)) : start=0, stop=문자 길이만큼 주어진 경우로 0을 시작으로 1 간격으로 떨어진 수들을 [0,1,…,문자길이수]의 형태로 반환함을 의미

Data Viewing

데이터가 어떻게 확산되는지 살펴보겠음

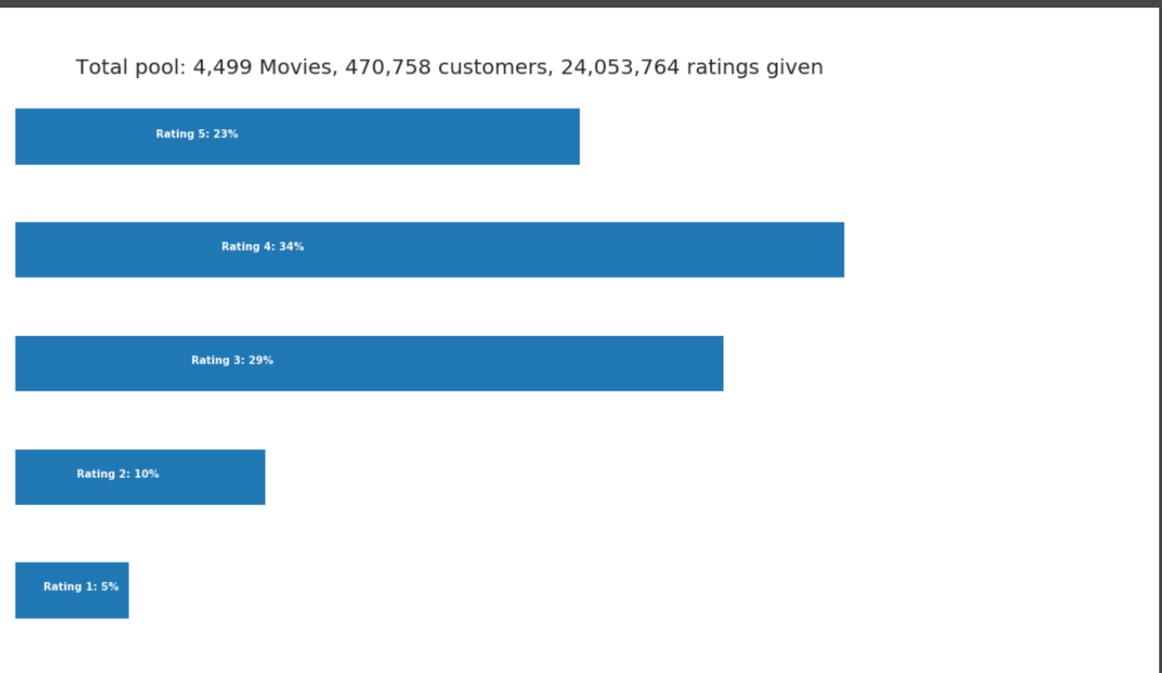
① agg() : 여러 개의 함수를 여러 열에 적용하는 함수

② isnull() : 관측치가 결측이면 True, 결측이 아니면 False의 boollean 값을 반환

③ nunique() : 총 고유값 수가 몇 개인지 알고 싶을 때
* unique() : 고유값이 무엇이 있는지 알고 싶을 때

▶ 평점이 상대적으로 긍정적(>3)인 경향이 있음
이는 불만족 고객은 보통 평가조차 하지 않기 때문
따라서, 낮은 등급의 영화는 일반적으로 정말 별로인
영화라는 것을 의미함

```
p = df.groupby('Rating')['Rating'].agg(['count'])

# get movie count
movie_count = df.isnull().sum()[1]

# get customer count
cust_count = df['Cust_Id'].nunique() - movie_count

# get rating count
rating_count = df['Cust_Id'].count() - movie_count

ax = p.plot(kind = 'barh', legend = False, figsize = (15,10))
plt.title('Total pool: {:,} Movies, {:,} customers, {:,} ratings given'.format(movie_coun
t, cust_count, rating_count), fontsize=20)
plt.axis('off')

for i in range(1,6):
    ax.text(p.iloc[i-1][0]/4, i-1, 'Rating {}: {:.0f}%'.format(i, p.iloc[i-1][0]*100 / p.s
um()[0]), color = 'white', weight = 'bold')
```



Data Cleaning

```
df_nan = pd.DataFrame(pd.isnull(df.Rating))
df_nan = df_nan[df_nan['Rating'] == True]
df_nan = df_nan.reset_index()

movie_np = []
movie_id = 1

for i,j in zip(df_nan['index'][1:],df_nan['index'][:-1]):
    # numpy approach
    temp = np.full((1,i-j-1), movie_id)
    movie_np = np.append(movie_np, temp)
    movie_id += 1

# Account for last record and corresponding length
# numpy approach
last_record = np.full((1,len(df) - df_nan.iloc[-1, 0] - 1),movie_id)
movie_np = np.append(movie_np, last_record)

print('Movie numpy: {}'.format(movie_np))
print('Length: {}'.format(len(movie_np)))
```

```
Movie numpy: [1.000e+00 1.000e+00 1.000e+00 ... 4.499e+03 4.499e+03 4.499e+03]
Length: 24053764
```

영화ID 데이터는 정말 복잡하고 엉망인 상태
영화ID 열을 추가하기 위해 데이터 프레임을 살펴보면 너무 비효율적이므로 커널의 메모리가 부족해짐
따라서 numpy 배열을 만든 다음 전체 배열을 기본 데이터 프레임에 열로 추가하여 작업을 수행함

① Numpy : 다차원 배열(행렬)을 효과적으로 처리하며 메모리의 효율성도 리스트보다 앞서게 됨

② temp : 두 변수 값을 맞바꾸기 위한 변수
ex. a=10, b=20
temp = a, a=b, b=temp
print(a,b) = 20 10

Data Cleaning

```
# remove those Movie ID rows
df = df[pd.notnull(df['Rating'])]

df['Movie_Id'] = movie_np.astype(int)
df['Cust_Id'] = df['Cust_Id'].astype(int)
print('-Dataset examples-')
print(df.iloc[::5000000, :])
```

```
-Dataset examples-
         Cust_Id  Rating  Movie_Id
1        1488844     3.0         1
5000996   501954     2.0       996
10001962  404654     5.0      1962
15002876  886608     2.0      2876
20003825 1193835     2.0      3825


/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  """
```

영화ID 열 제거

① notnull : 결측값을 False로, 결측값이 아닌 값을 True로 반환 *isnull함수는 반대

② astype(int) : '정수 ' 형태로 저장하겠다는 의미

numpy에 의해 Movie_Id가 정수형태의 숫자로 바뀐 것을 알 수 있음

Data Slicing

데이터 세트가 엄청나게 커졌기 때문에 데이터 품질을 개선하여 데이터 볼륨을 줄이려고 함

- 리뷰가 너무 적은 영화 삭제(상대적으로 인기 없음)
- 너무 적은 리뷰를 제공하는 고객 삭제(비교적 활동 적음)

위 벤치마크를 사용하면 인기없는 영화와 비활성 고객이 매트릭스 관점에서 여전히 동일한 볼륨을 차지하므로 효율성이 크게 향상되고(NaN은 여전히 공간 차지) 통계적 의미도 향상시키는 데 도움이 될 것임

① map() : 여러 개의 데이터를 한 번에 다른 형태로 변환
map(int) : 정수 형태로 변환
② round(실수,n) : 소수점을 n번째 까지만 표현하고 반올림
③ quantile() : pandas의 백분위수 구하는 함수
*numpy의 경우 percentile() 활용
④ benchmark : 두 함수의 성능(속도)를 비교할 때 활용
*pytest-benchmark

```
f = ['count','mean']

df_movie_summary = df.groupby('Movie_Id')['Rating'].agg(f)
df_movie_summary.index = df_movie_summary.index.map(int)
movie_benchmark = round(df_movie_summary['count'].quantile(0.
7),0)
drop_movie_list = df_movie_summary[df_movie_summary['count'] <
movie_benchmark].index

print('Movie minimum times of review: {}'.format(movie_benchmar
k))

df_cust_summary = df.groupby('Cust_Id')['Rating'].agg(f)
df_cust_summary.index = df_cust_summary.index.map(int)
cust_benchmark = round(df_cust_summary['count'].quantile(0.7),
0)
drop_cust_list = df_cust_summary[df_cust_summary['count'] < cus
t_benchmark].index

print('Customer minimum times of review: {}'.format(cust_benchm
ark))
```

```
Movie minimum times of review: 1799.0
Customer minimum times of review: 52.0
```

Data Slicing

Trim Shape(다듬어진 형태)를 보면 데이터 크기가 감소한 것을 알 수 있음

① isin() : 특정값이 데이터프레임에 있는지 확인
② df.shape : 데이터프레임의 행렬 개수 얻기

이제 추천 시스템을 위하여 데이터 세트를 피벗 테이블 형태(행렬)로 만들어보겠음

③ pivot_table()
index : 행 위치에 들어갈 열
columns : 열 위치에 들어갈 열
values : 데이터로 사용할 열

```
print('Original Shape: {}'.format(df.shape))
df = df[~df['Movie_Id'].isin(drop_movie_list)]
df = df[~df['Cust_Id'].isin(drop_cust_list)]
print('After Trim Shape: {}'.format(df.shape))
print('-Data Examples-')
print(df.iloc[::5000000, :])
```

```
Original Shape: (24053764, 3)
After Trim Shape: (17337458, 3)
-Data Examples-
          Cust_Id  Rating  Movie_Id
696        712664     5.0         3
6932490   1299309     5.0      1384
13860273   400155     3.0      2660
20766530   466962     4.0      3923
```

```
df_p = pd.pivot_table(df,values='Rating',index='Cust_Id',column
s='Movie_Id')

print(df_p.shape)
```

```
(143458, 1350)
```

Data Mapping

기존 프레임을 새로운 데이터 프레임으로 수정하려면 **inplace=True**를 넣어주면 된다. (딘

```
df_title = pd.read_csv('../input/movie_titles.csv', encoding =
"ISO-8859-1", header = None, names = ['Movie_Id', 'Year', 'Nam
e'])
df_title.set_index('Movie_Id', inplace = True)
print (df_title.head(10))
```

①

Movie_Id	Year	Name
1	2003.0	Dinosaur Planet
2	2004.0	Isle of Man TT 2004 Review
3	1997.0	Character
4	1994.0	Paula Abdul's Get Up & Dance
5	2004.0	The Rise and Fall of ECW
6	1997.0	Sick
7	1992.0	8 Man
8	2004.0	What the #$*! Do We Know!?
9	1991.0	Class of Nuke 'Em High 2
10	2001.0	Fighter

영화 매핑 파일 가져오기

① inplace = True : 기존 프레임을 새로운 데이터프레임으로 수정

영화 ID / 개봉 년도 / 영화제목 나타냄

03

추천 모델

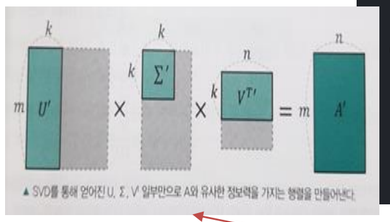
Recommendation models

필요한 모든 데이터를 로드하고 정리했으니, 이제 추천 시스템으로 들어가 보겠음

Recommend with Collaborative Filtering

```
reader = Reader()

# get just top 100K rows for faster run time
data = Dataset.load_from_df(df[['Cust_Id', 'Movie_Id', 'Rating']][:], reader)
#data.split(n_folds=3)

svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'])
```

```
{'test_rmse': array([0.8482087 , 0.84889465, 0.8481657 , 0.8474948 , 0.84746133]),
 'test_mae': array([0.65852945, 0.65868043, 0.65812747, 0.65800946, 0.65789018]),
 'fit_time': (1782.2070398330688,
  1785.608316898346,
  1780.5715522766113,
  1795.4553835391998,
  1797.4955141544342),
 'test_time': (141.48689699172974,
  128.33869457244873,
  138.72104907035828,
  137.38773250579834,
  116.30793190002441)}
```



▲ SVD를 통해 얻어진 U, Σ, V 일부만으로 A와 유사한 정보력을 가지는 행렬을 만들어낸다.

① SVD : 특이값 분해, A라는 mxn의 행렬을 U, Σ ,v^t 라는 3개의 행렬로 분해
(차원 축소 기법)

```
df_785314 = df[(df['Cust_Id'] == 785314) & (df['Rating'] == 5)]
df_785314 = df_785314.set_index('Movie_Id')
df_785314 = df_785314.join(df_title)['Name']
print(df_785314)
```

Movie_Id	
57	Richard III
175	Reservoir Dogs
311	Ed Wood
329	Dogma
331	Chasing Amy
395	Captain Blood
788	Clerks
798	Jaws
907	Animal Crackers
985	The Mummy

① 유저 785314가 과거에 좋아한 영화 ID와 타이틀명

Recommend with Collaborative Filtering

```
user_785314 = df_title.copy()
user_785314 = user_785314.reset_index()
user_785314 = user_785314[~user_785314['Movie_Id'].isin(drop_mo
vie_list)]

# getting full dataset
data = Dataset.load_from_df(df[['Cust_Id', 'Movie_Id', 'Ratin
g']], reader)

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

user_785314['Estimate_Score'] = user_785314['Movie_Id'].apply(l
ambda x: svd.predict(785314, x).est)

user_785314 = user_785314.drop('Movie_Id', axis = 1)

user_785314 = user_785314.sort_values('Estimate_Score', ascendi
ng=False)
print(user_785314.head(10))
```

	Year	Name	Estimate_Score
3167	1987.0	Evil Dead 2: Dead by Dawn	5.000000
1740	1999.0	Cowboy Bebop Remix	4.915521
1688	2003.0	Concert for George	4.836479
2113	2002.0	Firefly	4.835647
1551	1983.0	Black Adder	4.810998
871	1954.0	Seven Samurai	4.806245
4391	1993.0	Army of Darkness	4.744120
1031	1992.0	Hard Boiled	4.739616
1617	1984.0	Nausicaa of the Valley of the Wind	4.731034
2464	1984.0	This Is Spinal Tap	4.715362

유저 785314가 좋아할 영화 예측

① reset_index() : 컬럼명 인덱스가 아닌 행 번호 인덱스로 사용(컬럼명 지정한 인덱스 제거 가능)

② trainset : training 과정에서 모델이 잘 학습이 되는지 확인하고 검증(데이터셋 분할)

③ lambda(익명함수) : 정의해두고 쓰는 것이 아닌 필요한 곳에서 즉시 사용하고 버릴 수 있는 함수

④ ascending=False : ascending(오름차순)이 False이므로, 내림차순으로 정렬

Recommend with Pearsons'R correlations

```
def recommend(movie_title, min_count):
    print("For movie ({})".format(movie_title))
    print("- Top 10 movies recommended based on Pea
rsons'R correlation - ")
    i = int(df_title.index[df_title['Name'] == movi
e_title][0])
    target = df_p[i]
    similar_to_target = df_p.corrwith(target)
    corr_target = pd.DataFrame(similar_to_target, c
olumns = ['PearsonR'])
    corr_target.dropna(inplace = True)
    corr_target = corr_target.sort_values('Pearson
R', ascending = False)
    corr_target.index = corr_target.index.map(int)
    corr_target = corr_target.join(df_title).join(d
f_movie_summary)[['PearsonR', 'Name', 'count', 'mea
n']]
    print(corr_target[corr_target['count']>min_coun
t][:10].to_string(index=False))
```

① ② ③

```
recommend("What the #$*! Do We Know!?", 0)
```

```
For movie (What the #$*! Do We Know!?)
- Top 10 movies recommended based on Pearsons'R correlation -
 PearsonR                                  Name  count      mean
 1.000000                What the #$*! Do We Know!?  14910  3.189805
 0.505500                                Inu-Yasha   1883  4.554434
 0.452807  Captain Pantoja and the Special Services   1801  3.417546
 0.442354                  Without a Trace: Season 1   2124  3.980226
 0.384179                     Yu-Gi-Oh!: The Movie   3173  3.331547
 0.383959                                 Scorched   2430  2.894239
 0.381173   All Creatures Great and Small: Series 1   2327  3.938118
 0.381112          As Time Goes By: Series 1 and 2   2249  4.164073
 0.373018                      Cowboys & Angels   2368  3.589527
 0.371981                      Biggie & Tupac   1866  3.019293
```

```
recommend("X2: X-Men United", 0)
```

```
For movie (X2: X-Men United)
- Top 10 movies recommended based on Pearsons'R correlation -
 PearsonR                                  Name  count      mean
 1.000000                          X2: X-Men United  98720  3.932202
 0.384550                Batman Beyond: The Movie   2614  3.726855
 0.375967                          Justice League   3591  3.710944
 0.361393          Justice League: Justice on Trial   2961  3.718001
 0.338025          Batman Beyond: Return of the Joker   3704  3.604752
 0.335256                          Batman Begins  54922  4.236699
 0.328229            Batman: Mask of the Phantasm   2823  3.767977
 0.327040  Batman: The Animated Series: Tales of the Dark...   2432  3.583059
 0.316666            Dragon Ball Z: Super Android 13   2426  3.428689
 0.316166                Mortal Kombat: The Movie   7633  3.165466
```

피어슨 R 상관관계를 사용하여 모든 영화 쌍의 리뷰 점수 간 선형 상관관계를 측정한 다음 상관관계가 가장 높은 상위 10개 영화를 제공함

*피어슨 상관계수: 두 변수간의 관련성을 구하기 위해 이용 (r값은 x와y가 완전히 동일하면 1, 전혀 다르면 0, 1에 가까울수록 양의 상관관계를 가짐)

① p.corrwith(x['d']) : 'd'변수와 나머지 모든 변수 간 그룹 별 상관계수 구하는 것으로 'target'변수와 나머지 모든 변수간 관련성 비교

② dropna : 결측치 제거

③ inplace=True : 원본 데이터 프레임 보존할 필요없이 바로 결측치 제거해서 수정하고 싶을 때

THANK YOU

감사합니다.