

Python Programming and Practice

Movie recommendation system

Progress Report #1

Date : 2023.11.26

Name : Hyeonseung Lee

ID : 236179

1. Introduction

1) Background

Until recently, so many movies have been released, and so many movies have been piled up from existing films to current ones. This makes it difficult for people to choose which movies they want to watch. Also, I hope that once the movie chosen is fun.

2) Project goal

It aims to create a system that recommends products that users may like by analyzing the user's past viewing history and actual stars left by the user.

3) Differences from existing programs

Existing programs can analyze the user's viewing history, find similar titles according to the title entered by the user, or if they only recommend similarities such as genres and directors, they can predict how many ratings users will give when they see additional movies they currently recommend.

2. Functional Requirement

1) Function 1

- A movie that user like

(1) Detailed function 1

- Analyzing and saving the user's viewing history, viewing time, etc

(2) Detailed function 2

- Analyzing the ratings left by the user or the ratings of others among movies watched by the user

2) Function 2

- The ability to find similar movies

(1) Detailed function 1

- Find a similar movie among the movies that the user believes they like. (genre, director, plot, etc.)

(2) Detailed function 2

- Rank the movie in order that the user will like and expose the top 5 to 10 movies.

3) Function 3

- Predicting user ratings

(1) Detailed function 1

- Predict the satisfaction level (score) of the recommended movie and display it with the movie recommendation list

3. Progress

1) Implementation of features

(1) Feature Name Implemented

1. Do_apriori function in the movie_recommender.py file: This function takes a list of movies selected by the user, movie datasets, and evaluation datasets as input, and generates a list of movies to recommend using the Apriori algorithm

- Input and output

· Input: movie list (_input_movies), movie dataset (_movies_df), evaluation dataset (_ratings_df)

· Output: List of recommended movies generated by the Apriori algorithm (_apriori_result)

- Applied learning: functions, conditional statements, repetitive statements, modules

- Code Screenshot

```
def do_apriori(_input_movies, _movies_df, _ratings_df):
    # Internal variables
    _apriori_result = []

    """ Remove the Nan title & join the dataset """
    Nan_title = _movies_df['title'].isna()
    _movies_df = _movies_df.loc[Nan_title == False]

    _movies_df = _movies_df.astype({'id' : 'int64'})
    df = pd.merge(_ratings_df, _movies_df[['id', 'title']], left_on='movieId', right_on='id')
    df.drop(['timestamp', 'id'], axis=1, inplace=True)

    """ Prepare Apriori
    |   row : userId | col : movies   """
    df = df.drop_duplicates(['userId', 'title'])
    df_pivot = df.pivot(index='userId', columns='title', values='rating').fillna(0)
    df_pivot = df_pivot.astype('int64')
    df_pivot = df_pivot.applymap(apriori_encoding)
    # print(df_pivot.head())
```

```

""" A-priori Algorithm """
#calculate support and eradicate under min_support
frequent_items = apriori(df_pivot, min_support=0.07, use_colnames=True)
# print(frequent_items.head())

# using association rules, compute the other parameter ex) confidence, lift ..
association_indicator = association_rules(frequent_items, metric="lift", min_threshold=1)

# sort by order of lift
df_lift = association_indicator.sort_values(by=['lift'], ascending=False)
# print(df_res.head())

""" Start recommendation """
for selected_movie in _input_movies:
    num = 0
    df_selected = df_lift[df_lift['antecedents'].apply(lambda x: len(x) == 1 and next(iter(x)) == selected_movie)]
    df_selected = df_selected[df_selected['lift'] > 1.2]
    recommended_movies = df_selected['consequents'].values

    for movie in recommended_movies:
        for title in movie:
            if title not in _apriori_result and num < 10:
                _apriori_result.append(title)
                num += 1

return _apriori_result

```

2. Do_means function in the movie_recommender.py file: This function takes a list of recommended movies generated through the Apriori algorithm, a list of movies selected by the user, and a movie dataset as input to generate a final list of recommended movies using K-means clustering and visualizes them as pylabs

- Input and output

- Input: A list of recommended movies (_apriori_result) generated by the Apriori algorithm, a list of movies selected by the user (_input_movies), a movie dataset (_movies_df)

- Output: A list of final recommended movies (_kmeans_result) generated by K-means clustering

- Applied learning: functions, conditional statements, repetitive statements, modules

- Code Screenshot

```
def do_kmeans(_apriori_result, _input_movies, _movies_df):
    # record all clusters in _input_movies
    clusters = []
    _kmeans_result = []

    numeric_df = _movies_df[['budget', 'popularity', 'revenue', 'runtime', 'vote_average', 'vote_count', 'title']]

    numeric_df.isnull().sum()
    numeric_df.dropna(inplace=True)
    # print(df_numeric['vote_count'].describe())

    """cut off the movies' votes less than 25"""
    df_numeric = numeric_df[numeric_df['vote_count'] > 25]

    # Normalize data - by MinMax scaling provided by sklearn
    minmax_processed = preprocessing.MinMaxScaler().fit_transform(df_numeric.drop('title', axis=1))
    df_numeric_scaled = pd.DataFrame(minmax_processed, index=df_numeric.index, columns=df_numeric.columns[:-1])

    """Apply K-means clustering"""
    # make elbow curve to determine value 'k'
    num_cluster = range(1, 20)
    kmeans = [KMeans(n_clusters=i) for i in num_cluster]
    score = [kmeans[i].fit(df_numeric_scaled).score(df_numeric_scaled) for i in range(len(kmeans))]
```

```
# print elbow curve
pl.plot(num_cluster, score)
pl.xlabel("Number of clusters")
pl.ylabel("Score")
pl.title("Elbow curve")
#plt.show() # maybe k=4 is appropriate

# Fit K-means clustering for k=5
kmeans = KMeans(n_clusters=5)
kmeans.fit(df_numeric_scaled) # result is kmeans_label

# write back labels to the original numeric data frame
df_numeric['cluster'] = kmeans.labels_
# print(df_numeric.head())

# Search all clusters in user selected movies
for movie1 in _input_movies:
    try:
        cluster_candid = df_numeric.loc[df_numeric["title"] == movie1, 'cluster'].values[0]
        # print(cluster_candid)
        clusters.append(cluster_candid)
    except IndexError as e:
        msg = "There is No cluster in movie [" + movie1 + "]"
        ErrorLog(msg)
        #print(msg)
```

```

# Filtering movies that are not in clusters
for movie2 in _apriori_result:
    try:
        cluster_tmp = df_numeric.loc[df_numeric["title"] == movie2, 'cluster'].values[0]
        if cluster_tmp in clusters:
            _kmeans_result.append(movie2)
    except IndexError as e:
        msg = "There is No cluster in movie [" + movie2 + ']'
        ErrorLog(msg)
        #print(msg)

return _kmeans_result

```

3. Module implementation part of main.py file: main code of the project that loads data using the pd.read_csv function, reads the csv file, reads the csv file, and preprocesses the data, executes the recommended system, outputs and stores the results

- Input and output

· Input: List of movies selected by the user (input_movies)

· Output: recommended movie list (final_result), on-screen message, message stored in result.txt file

- Applied Learning: Functions, Conditional Statements, Modules

- Code Screenshot

```

import pandas as pd
import warnings; warnings.simplefilter('ignore')
from movie_recommender import *
from data_reader import *

def main(input_movies):
    final_result = ""
    final_result += "Selected movies (5 movies) : " + ",".join(input_movies) + "\n\n"
    print(final_result)

    # csv파일을 읽어온다
    movies_df = pd.read_csv('C:/Users/user/Desktop/현승/23_1학기_학교/C 및 PY\PY_프로젝트/PY202309-P/Sources/movie_dataset/movies_metadata.csv')
    ratings_df = pd.read_csv('C:/Users/user/Desktop/현승/23_1학기_학교/C 및 PY\PY_프로젝트/PY202309-P/Sources/movie_dataset/ratings_small.csv')

    # Drop the trash(error) data
    movies_df = drop_trash_data(movies_df)

    # a-priori & k-means를 이용해서 추천을 해준다
    apriori_result = do_apriori(input_movies, movies_df, ratings_df)
    kmeans_result = do_kmeans(apriori_result, input_movies, movies_df)

```

```

# 추가해야될 부분
# kmeans_result를 문자열로 변환
# 영화 목록을 심표로 구분
# 저장

# 결과를 result.txt파일로 출력해줌
print(final_result)
f = open("result.txt", "w")
f.write(final_result)
f.close()

return final_result

if __name__ == '__main__':
    #미구현 대체 : input_movies를 임시로 지정
    input_movies = ['Toy Story', 'Jumanji', 'Grumpier Old Men', 'Waiting to Exhale', 'Father of the Bride Part II']
    main(input_movies)

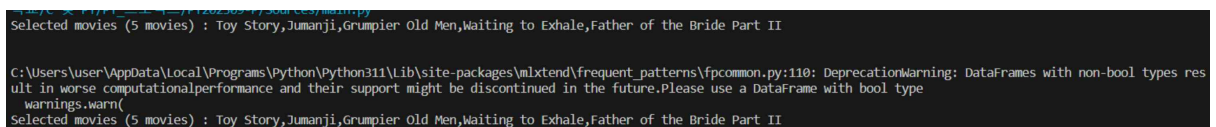
```

2) Test Results

(1) Pre-processing datasets and creating a list of recommended movies

- Roles to load required data and perform data preprocessing
- The role of finding movies similar to those of your choice using two recommended algorithms, do_apriori and do_kmeans

- Test Results Screenshot



```

C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types res
ult in worse computational performance and their support might be discontinued in the future.Please use a DataFrame with bool type
warnings.warn(
Selected movies (5 movies) : Toy Story,Jumanji,Grumpier Old Men,Waiting to Exhale,Father of the Bride Part II

```

- Warning that using non-boolean type data frames can degrade computational performance
- Currently, it is difficult to judge the exact output because other codes are not applied, but it is doubtful that the recommendation system is working properly as it shows the same output compared to the temporarily entered user's movies

4. Changes in Comparison to the Plan

- None

5. Schedule

Work		11/3	11/6~12	11/20~30	12/1~12/9
Create a proposal		완료			
Function 1	Detailed function 1		완료		
	Detailed function 2			진행중	
Function 2	Detailed function 1				----->

Work		12/10~14	12/15~22
Function 2	Detailed function 2	----->			
Function 3	Detailed function 1		----->		