Python Programming and Practice

Movie recommendation system

Progress Report #2

Date :	2023.12.10
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. First, remove the NaN title and combine the dataset. To prepare the Apriori algorithm, transform data frames, calculate frequency items, and remove items with a minimum support level. Use association rules to calculate other parameters (e.g., reliability, lift, etc.). Find a movie associated with the movie selected by the user and create a list of recommended movies. In this case, the movie is excluded from the recommended list, and when the number of recommended movies reaches 5, the recommendation is stopped.

- Input and output

 · Input: List of movies selected by the user (_input_movies), movie dataset (_movies_df), evaluation dataset (_ratings_df)

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

- Applied learning: Function, Conditional Statements, Loop, Exception

- Code Screenshot

```python
def do_apriori(_input_movies, _movies_df, _ratings_df):
    try:
        # 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)

        # Check if df is empty
        print(f"df shape: {df.shape}")
        if df.empty:
            print("df is empty")
            return _apriori_result

        """ 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).astype(bool)
        # print(df_pivot.head())
```

```python
# Check if df_pivot is empty
print(f"df_pivot shape: {df_pivot.shape}")
if df_pivot.empty:
    print("df_pivot is empty")
    return _apriori_result

""" 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())

# Check if frequent_items is empty
print(f"frequent_items shape: {frequent_items.shape}")
if frequent_items.empty:
    print("frequent_items is empty")
    return _apriori_result

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

```python
# Check if association_indicator is empty
print(f"association_indicator shape: {association_indicator.shape}")
if association_indicator.empty:
    print("association_indicator is empty")
    return _apriori_result

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

""" Start recommendation """
for r in range(len(_input_movies), 0, -1):
    for selected_movies in combinations(_input_movies, r):
        df_selected = df_lift[df_lift['antecedents'].apply(lambda x: set(x) == set(selected_movies))]
        df_selected = df_selected[df_selected['lift'] > 1.0]
        df_selected.sort_values(by='lift', ascending=False, inplace=True)  # Sort by lift in descending order

        recommended_movies = df_selected['consequents'].values

        for movie in recommended_movies:
            for title in movie:
                if title not in _input_movies and title not in _apriori_result:
                    _apriori_result.append(title)
                    if len(_apriori_result) == 5:  # Stop when 5 movies are recommended
                        break
```

```python
                if len(_apriori_result) == 5:
                    break
            if len(_apriori_result) == 5:
                break

    except Exception as e:
        print(f"Error in do_apriori: {e}")

    return _apriori_result
```

2) kmeans similarity analysis algorithm

1. The recommended movie list (_apriori_result), the user's selected movie list (_input_movies), and the movie dataset (_movies_df) received from the Apriori algorithm are received as inputs, and the recommended movie list is filtered using the K-Means clustering algorithm.

- Input and output

 · Input: A list of recommended movies (_apriori_result) received from the Apriori algorithm, a list of movies selected by the user (_input_movies), and a movie dataset (_movies_df)

 · Output: List of recommended movies filtered by the K-Means clustering algorithm (_kmeans_result)

- Applied learning: Function, Conditional Statements, Loop, Exception

-CodeScreenshot

```
def do_kmeans(_apriori_result, _input_movies, _movies_df):
    try:
        # 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])
```

```python
"""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())
```

```python
        # 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)
    except Exception as e:
        print(f"Error in do_kmeans: {e}")

    return _kmeans_result
```

2) Test Reslults

(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

```
PS C:\Users\user\Desktop\현승\23_1학기_학교\C 및 PY\PY_프로젝트\PY202309-P\Sources> python -u "c:\Users\user\Desktop\
현승\23_1학기_학교\C 및 PY\PY_프로젝트\PY202309-P\Sources\main.py"
마음에 들었던 영화를 입력하세요:
Selected movies (5 movies) : Toy Story,Jumanji,Grumpier Old Men,Transformers,Batman Forever

df shape: (44994, 4)
df_pivot shape: (671, 2794)
    support                    itemsets
0  0.129657  (20,000 Leagues Under the Sea)
1  0.129657         (2001: A Space Odyssey)
2  0.298063                     (48 Hrs.)
3  0.292101                 (5 Card Stud)
4  0.093890       (A Brief History of Time)
frequent_items shape: (91008, 2)
association_indicator shape: (2569674, 10)
```

```
Apriori recommendations:


K-means recommendations:
```

Originally, the part where the user receives a list of five movies was inputted from the code and replaced, and the size of the data frame is outputted for testing,
5 data frames were output using pandas (must be corrected so that the top 5 can be output in order)

The bug needs to be fixed in areas that don't actually print out recommended 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				진행중

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