Python Programming and Practice

Movie recommendation system

Final Report

Date : 2023.12.24
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. Implementation

(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

```
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())

        # 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=0.1, 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)

# 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]
        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

                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])

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

return _kmeans_result
```

4. Test Result

(1) do apriori

- This code is an implementation of a movie recommendation system using the A-priori algorithm. The main workflow is as follows:

1. Data preprocessing: First, remove the NaN value, convert the data type appropriately. After that, merge the ratings data and the movies data, and remove unnecessary columns.

2. Prepare Apriori: After deduplication with 'userId' and 'tile', create a pivot table, which indicates whether each user has rated each movie.

3. Apply A-priori algorithm: Use the apriori function to find frequent item sets with minimum support (min_support) of 0.1 or more. And use the association_rules function to find association rules between frequent item sets.

4. Start Recommendations: Create a combination of the list of movies you have entered and find associated rules that include them. Sort rules with a lift value of 1 or more in descending order, and find and add movies that have not yet been recommended to your list of recommendations. When you have five recommended movies, end the recommendation.

5. Dealing with exceptions: If an error occurs in the middle, print it out and return a blank list of recommendations.

This code implements a system that recommends other movies that are highly relevant to it, based on the movies that the user has already watched. The A-priori algorithm is often used in these item-based recommendation systems.

- test reuslt screenshot

```
Apriori recommendations:
The Hours, Terminator 3: Rise of the Machines, Back to the Future Part II, Rain Man, Cockles and Muscles, Sissi, The Conversation, Ba
ng, Boom, Bang, Monsoon Wedding, Romeo + Juliet, The Science of Sleep, Three Colors: Red, Solaris, Reservoir Dogs, The Million Dollar
 Hotel, A Nightmare on Elm Street, Titanic, Grill Point, Tough Enough, Batman Returns, Live and Let Die, To Kill a Mockingbird, Dave
Chappelle's Block Party, Syriana, Big Fish, Lost in Translation, The Passion of Joan of Arc, Silent Hill, Psycho, Ocean's Eleven, Onc
e Were Warriors, Beauty and the Beast, Men in Black II, A Clockwork Orange, Wag the Dog, Jurassic Park, The 39 Steps, All the Way Boy
s, Rope, Say Anything..., Young and Innocent, 5 Card Stud
```

(2) kmeans similarity analysis algorithm

This code is an implementation of a movie recommendation system using the K-means clustering algorithm. The main workflow is as follows:

1. Data preprocessing: First, we extract only numeric characteristics from movie data, and then we remove missing values. Then, we only leave movies with 'vote_count' exceeding 25.

Normalize data: Use sklearn's MinMaxScaler to normalize data.

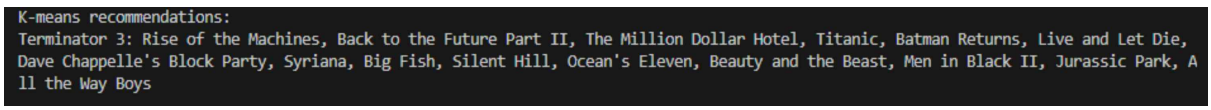
2. Apply K-means clustering: First, find the optimal number of clusters using the Elbow method. Then, apply the K-means algorithm to divide the films into clusters.

Start Recommendation: Find the cluster to which the movies you have entered belong. And, add only the movies that belong to this cluster to the list of recommendations among the movies recommended by the A-3. priori algorithm.

Dealing with exceptions: If an error occurs in the middle, print it out and return a blank list of recommendations.

This code implements a system that recommends other movies with similar characteristics to the movies that users have already watched. K-means clustering is often used in these item-based recommendation systems.

- test reuslt screenshot



5. Changes in Comparison to the Plan

1) GPA PREDICTION SYSTEM

- Previously: a system that predicts and informs the ratings of movies recommended to match the ratings of previous users through sklearn
- Later: Delete
- Reason:

    1. Absence of a user registration system
    2. Need information on user ratings of your previous movies
    3. The amount of data set in the measurement evaluation information of the user's previous movies is too small

4. Unlearned

6. Lessons Learned & Feedback

It was a good time to review and learn how to use by myself by coding the project using what I learned in class, but when I did the program-making project for the first time, I felt that it was not as easy as I thought