Movie Recommendation

August 3, 2024

1 BIG DATA CAPSTONE PROJECT : MILESTONE REPORT 4

2 PROFESSOR: MEYSAM EFFATI

3 TEAM: ARUN KUMAR SUBRAMANIAM, CARLOS MUNOZ EBRATT, KAMALDEEP KAUR, LUIS ALEJANDRO GUTIERREZ HAYEK, OSCAR FELIPE FERNANDEZ TOVAR

3.0.1 Importing the libraries

[1]:
```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

3.0.2 Loading the dataset

[2]:
```python
movies_df = pd.read_csv('mflix.movies.csv')
```

3.0.3 Data Exploration

The df.head() function in pandas is used to display the first few rows of a DataFramef. By default, it shows the first 5 rows.

[3]:
```python
print(movies_df.head())
```

```
                                                plot  genres[0] genres[1]  \
0  The cartoonist, Winsor McCay, brings the Dinos…  Animation     Short
1  An immigrant leaves his sweetheart in Italy to…      Drama       NaN
2  A rich young Easterner who has always wanted t…     Comedy   Western
3  A penniless young man tries to save an heiress…     Comedy     Short
4  A tipsy doctor encounters his patient sleepwal…     Comedy     Short

  genres[2]  runtime        cast[0]        cast[1]          cast[2]  \
0    Comedy     12.0   Winsor McCay  George McManus  Roy L. McCardell
1       NaN     78.0   George Beban  Clara Williams    J. Frank Burke
```

```
2   Romance     72.0  Douglas Fairbanks    Eileen Percy    Calvert Carter
3    Action     22.0       Harold Lloyd  Mildred Davis    'Snub' Pollard
4       NaN     26.0       Harold Lloyd      Roy Brooks     Mildred Davis

            cast[3]  num_mflix_comments  …     awards.text lastupdated  year  \
0               NaN                   0  …          1 win.     03:15.3  1914
1       Leo Willis                    0  …          1 win.     07:43.2  1915
2   Charles Stevens                   0  …          1 win.     40:35.1  1917
3  Peggy Cartwright                   0  …  1 nomination.     16:14.2  1919
4       Wallace Howe                  1  …  1 nomination.     35:33.7  1920

  imdb.rating imdb.votes imdb.id countries[0]   type tomatoes.viewer.rating  \
0         7.3     1837.0    4008          USA  movie                    3.7
1         6.4      175.0    5557          USA  movie                    4.0
2         6.9      388.0    8775          USA  movie                    NaN
3         7.0      639.0   10146          USA  movie                    3.3
4         7.0      646.0   11293          USA  movie                    3.4

    rated
0     NaN
1  PASSED
2     NaN
3    TV-G
4  PASSED

[5 rows x 31 columns]
```

The df.info() function in pandas provides a summary of a DataFrame df.

[4]:
```python
print(movies_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21349 entries, 0 to 21348
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   plot                    20203 non-null  object
 1   genres[0]               21237 non-null  object
 2   genres[1]               15345 non-null  object
 3   genres[2]               8696 non-null   object
 4   runtime                 20910 non-null  float64
 5   cast[0]                 20987 non-null  object
 6   cast[1]                 20686 non-null  object
 7   cast[2]                 20484 non-null  object
 8   cast[3]                 20309 non-null  object
 9   num_mflix_comments      21349 non-null  int64
 10  poster                  18044 non-null  object
 11  title                   21349 non-null  object
 12  fullplot                19852 non-null  object
```

```
 13  languages[0]           21119 non-null  object
 14  released               20878 non-null  object
 15  directors[0]           21107 non-null  object
 16  directors[1]           1580 non-null   object
 17  writers[0]             20256 non-null  object
 18  writers[1]             13427 non-null  object
 19  awards.wins            21349 non-null  int64
 20  awards.nominations     21349 non-null  int64
 21  awards.text            21349 non-null  object
 22  lastupdated            21349 non-null  object
 23  year                   21349 non-null  object
 24  imdb.rating            21288 non-null  float64
 25  imdb.votes             21287 non-null  float64
 26  imdb.id                21349 non-null  int64
 27  countries[0]           21339 non-null  object
 28  type                   21349 non-null  object
 29  tomatoes.viewer.rating 18566 non-null  float64
 30  rated                  11455 non-null  object
dtypes: float64(4), int64(4), object(23)
memory usage: 5.0+ MB
None
```

3.0.4 Data Handling

Getting the summary of missing values in each column

[5]:
```
missing_values = movies_df.isnull().sum()
missing_values = missing_values[missing_values > 0]
missing_values
```

[5]:
```
plot                    1146
genres[0]                112
genres[1]               6004
genres[2]              12653
runtime                  439
cast[0]                  362
cast[1]                  663
cast[2]                  865
cast[3]                 1040
poster                  3305
fullplot                1497
languages[0]             230
released                 471
directors[0]             242
directors[1]           19769
writers[0]              1093
writers[1]              7922
imdb.rating               61
```

```
imdb.votes                     62
countries[0]                   10
tomatoes.viewer.rating       2783
rated                        9894
dtype: int64
```

Dropping columns with excessive missing values

```
[6]: columns_to_drop = ['directors[1]', 'writers[1]', 'genres[2]']
     movies_df.drop(columns=columns_to_drop, inplace=True)
```

Merging genres and Cast into a single column

```
[7]: movies_df['Genre'] = movies_df[['genres[0]', 'genres[1]']].apply(lambda x: ', '.
       ↪join(x.dropna().astype(str)), axis=1)
```

```
[8]: movies_df['Cast'] = movies_df[['cast[0]', 'cast[1]', 'cast[2]', 'cast[3]']].
       ↪apply(lambda x: ', '.join(x.dropna().astype(str)), axis=1)
```

Dropping the columns that were merged

```
[9]: movies_df.drop(columns=['genres[0]', 'genres[1]', 'cast[0]', 'cast[1]',␣
       ↪'cast[2]', 'cast[3]'], inplace=True)
```

Removing rows with null values in specific columns because we can't fill the missing values with mode or media, since these are too unique.

```
[10]: columns_to_check = ['poster', 'fullplot', 'languages[0]', 'released',␣
        ↪'directors[0]','writers[0]', 'countries[0]','poster', 'awards.text',␣
        ↪'lastupdated', 'imdb.votes', 'type', 'rated']
      movies_df.dropna(subset=columns_to_check, inplace=True)
```

```
[11]: numerical_columns = ['runtime', 'imdb.rating', 'imdb.votes', 'tomatoes.viewer.
        ↪rating']
      for column in numerical_columns:
          movies_df[column].fillna(movies_df[column].median(), inplace=True)
```

Filling missing values in 'rated' with a placeholder

```
[12]: movies_df['rated'].fillna('Not Rated', inplace=True)
```

Checking for missing values

```
[13]: missing_values = movies_df.isnull().sum()
      missing_values = missing_values[missing_values > 0]
      missing_values
```

```
[13]: Series([], dtype: int64)
```

Renaming the columns with understandable names.

```
[14]: movies_df.rename(columns={
    'plot': 'Plot',
    'runtime': 'Runtime',
    'num_mflix_comments': 'Comments',
    'title': 'Title',
    'fullplot': 'FullPlot',
    'languages[0]': 'Language',
    'released': 'Released',
    'directors[0]': 'Director',
    'writers[0]': 'Writer',
    'awards.wins': 'Awards',
    'awards.nominations': 'Nominations',
    'year': 'Year',
    'imdb.rating': 'IMDB Rating',
    'countries[0]': 'Countries',
    'tomatoes.viewer.rating': 'Tomatoes Rating'
}, inplace=True)
```

Now checking the dataframe information

```
[15]: # Display the updated dataset information and first few rows
updated_info = movies_df.info()
updated_head = movies_df.head()
updated_info, updated_head
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10701 entries, 3 to 21317
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Plot             10701 non-null  object
 1   Runtime          10701 non-null  float64
 2   Comments         10701 non-null  int64
 3   poster           10701 non-null  object
 4   Title            10701 non-null  object
 5   FullPlot         10701 non-null  object
 6   Language         10701 non-null  object
 7   Released         10701 non-null  object
 8   Director         10701 non-null  object
 9   Writer           10701 non-null  object
 10  Awards           10701 non-null  int64
 11  Nominations      10701 non-null  int64
 12  awards.text      10701 non-null  object
 13  lastupdated      10701 non-null  object
 14  Year             10701 non-null  object
 15  IMDB Rating      10701 non-null  float64
 16  imdb.votes       10701 non-null  float64
 17  imdb.id          10701 non-null  int64
```

```
 18  Countries        10701 non-null  object
 19  type             10701 non-null  object
 20  Tomatoes Rating  10701 non-null  float64
 21  rated            10701 non-null  object
 22  Genre            10701 non-null  object
 23  Cast             10701 non-null  object
dtypes: float64(4), int64(4), object(16)
memory usage: 2.0+ MB
```

[15]:
```
(None,
                                                 Plot  Runtime  Comments  \
 3   A penniless young man tries to save an heiress…     22.0         0
 4   A tipsy doctor encounters his patient sleepwal…     26.0         1
 5   A young man, unaccustomed to children, must ac…     35.0         0
 9   Millie Stope lives with her grandfather on a r…     88.0         1
 11  Mrs Erlynne, the mother of Lady Windermere - h…    120.0         1

                                                poster                 Title  \
 3   https://m.media-amazon.com/images/M/MV5BNzE1OW…     From Hand to Mouth
 4   https://m.media-amazon.com/images/M/MV5BODliMj…         High and Dizzy
 5   https://m.media-amazon.com/images/M/MV5BYjgzYz…           Now or Never
 9   https://m.media-amazon.com/images/M/MV5BMjA4OT…           Wild Oranges
 11  https://m.media-amazon.com/images/M/MV5BZDUyYz…  Lady Windermere's Fan

                                             FullPlot Language  \
 3   As a penniless man worries about how he will m…  English
 4   After a long wait, a young doctor finally has …  English
 5   Mary is looking after a young child whose pare…  English
 9   Millie Stope lives with her grandfather on a r…  English
 11  Mrs Erlynne, the mother of Lady Windermere - h…  English

                     Released            Director                     Writer  \
 3   1919-12-28T00:00:00.000Z  Alfred J. Goulding      H.M. Walker (titles)
 4   1920-07-11T00:00:00.000Z           Hal Roach       Frank Terry (story)
 5   1921-03-27T00:00:00.000Z   Fred C. Newmeyer      H.M. Walker (titles)
 9   1924-01-20T00:00:00.000Z          King Vidor  Joseph Hergesheimer (by)
 11  1925-12-26T00:00:00.000Z      Ernst Lubitsch           Oscar Wilde (by)

     …  Year  IMDB Rating imdb.votes imdb.id Countries   type  \
 3   …  1919          7.0      639.0   10146       USA  movie
 4   …  1920          7.0      646.0   11293       USA  movie
 5   …  1921          6.8      489.0   12512       USA  movie
 9   …  1924          7.1      327.0   15498       USA  movie
 11  …  1925          7.6      630.0   16004       USA  movie

     Tomatoes Rating     rated           Genre  \
 3               3.3      TV-G   Comedy, Short
```

```
4               3.4    PASSED   Comedy, Short
5               3.8    PASSED   Comedy, Short
9               4.2    PASSED  Drama, Romance
11              3.7  APPROVED          Comedy

                                                 Cast
3   Harold Lloyd, Mildred Davis, 'Snub' Pollard, P…
4   Harold Lloyd, Roy Brooks, Mildred Davis, Walla…
5        Harold Lloyd, Mildred Davis, Anna Mae Bilson
9   Frank Mayo, Virginia Valli, Ford Sterling, Nig…
11  Ronald Colman, May McAvoy, Bert Lytell, Irene …

[5 rows x 24 columns])
```

3.0.5 Model 1

Vectorize the Genre column using TF-IDF

[16]:
```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(movies_df['Genre'])
```

Calculate the cosine similarity between the genre vectors

[17]:
```
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

Function to get movie recommendations based on genre similarity

[18]:
```
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = movies_df[movies_df['Title'].str.lower() == title.lower()].index[0]

    # Get the pairwise 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 indices of the 10 most similar movies
    sim_scores = sim_scores[1:11]  # Exclude the first movie (itself)

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

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

Example usage: Get recommendations for a specific movie

[19]:
```
recommendations = get_recommendations("A Walk in the Sun")
recommendations
```

[19]:
```
95                  Eskimo
113       The Emperor Jones
117             Comradeship
147         Dante's Inferno
155         Come and Get It
182            The Informer
269         Of Mice and Men
280      The Green Pastures
302             The Citadel
311                   Tevya
Name: Title, dtype: object
```

3.0.6 Model 2

Function to get movie recommendations based on genre and released year as input

Combine Genre and Released Year into a new feature

[20]:
```
movies_df['Genre_Released'] = movies_df['Genre'] + ' ' + movies_df['Released'].
 ↪astype(str)
```

Vectorize the combined Genre_Released column using TF-IDF

[21]:
```
tfidf_vectorizer_combined = TfidfVectorizer(max_features=500)
tfidf_matrix_combined = tfidf_vectorizer_combined.
 ↪fit_transform(movies_df['Genre_Released'])
```

[22]:
```
def recommend_movies_by_genre_year(genre, year,␣
 ↪tfidf_matrix=tfidf_matrix_combined, n_recommendations=10):
    # Combine genre and year into a single string
    input_combined = genre + ' ' + str(year)

    # Transform the input into a TF-IDF vector
    input_vector = tfidf_vectorizer_combined.transform([input_combined])

    # Calculate the cosine similarity of the input with all movies
    cosine_similarities = cosine_similarity(input_vector,␣
 ↪tfidf_matrix_combined).flatten()

    # Get the indices of the most similar movies
    sim_indices = cosine_similarities.argsort()[-(n_recommendations + 1):][::
 ↪-1][1:]

    # Return the top n most similar movies
    return movies_df['Title'].iloc[sim_indices]
```

Example usage: Get recommendations for a specific genre and release year

[23]:
```
recommended_movies = recommend_movies_by_genre_year("Action", 2012)
recommended_movies
```

[23]:
```
15755                       Bellflower
15815     The Man with the Iron Fists
17779                          The Day
17210                           My Way
17745              Here Comes the Boom
17362              Here Comes the Boom
16438                  Men in Black 3
17636                          Lockout
17179                          Lockout
18479                     Tai Chi Zero
Name: Title, dtype: object
```

3.0.7 Movie Recommendation System: Model Report

Introduction This project aims to build a movie recommendation system that provides movie suggestions based on the genre and release year input by a user. To achieve this, we have utilized a content-based filtering approach using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization and cosine similarity.

Models and Techniques Used

1. **TF-IDF Vectorization**:
    - **Purpose**: To convert textual data (genres and release years) into numerical vectors that can be used for similarity calculations.
    - **Why TF-IDF**: TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps in transforming the textual data into a matrix of features, capturing the relevance of terms in each movie description.
    - **Implementation**: The `TfidfVectorizer` from the `sklearn.feature_extraction.text` module was used with a maximum feature limit to manage memory usage.
2. **Cosine Similarity**:
    - **Purpose**: To calculate the similarity between the input movie (based on genre and release year) and all other movies in the dataset.
    - **Why Cosine Similarity**: Cosine similarity measures the cosine of the angle between two vectors, providing a measure of how similar the two vectors are irrespective of their magnitude. It is particularly useful for text data, as it helps to find the direction of the vectors (representing documents) rather than their lengths.
    - **Implementation**: The `cosine_similarity` function from the `sklearn.metrics.pairwise` module was used to compute the similarity between the TF-IDF vectors.

Steps in the Model

1. **Data Preparation**:
   - Combined the genre and release year of each movie into a single feature (`Genre_Released`) to provide a comprehensive representation of both attributes.
   - Example: "Drama 2000" combines the genre "Drama" with the release year "2000".
2. **Vectorization**:
   - Applied TF-IDF vectorization to the `Genre_Released` feature to convert the text data into a numerical matrix (`tfidf_matrix_combined`).
3. **Similarity Calculation**:
   - For a given input (genre and release year), transformed the input into a TF-IDF vector.
   - Calculated the cosine similarity between the input vector and all other movie vectors in the dataset.
4. **Recommendation Generation**:
   - Sorted the movies based on similarity scores and selected the top 10 most similar movieses = recommend_movies_by_genre_year("Drama", 2000) "'

Conclusion The content-based filtering approach using TF-IDF vectorization and cosine similarity effectively provides movie recommendations based on the genre and release year. This method leverages textual features to understand the context and similarity between movies, making it a robust solution for personalized movie recommendations.