Predictive Analytics: Assignment 10.2

Joshua Greenert

DSC630-T301 Predictive Analytics

2/14/2023

In [82]:
```
# Import the required libaries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Pull in the data to begin preparation.
df_ratings = pd.read_csv('../../../../../Downloads/ml-25m/ratings.csv')
df_ratings.head(5)
```

Out[82]:

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510

In [83]:
```
# Pull in the links, movies, and tags datasets to use later.
df_movies = pd.read_csv('../../../../../Downloads/ml-25m/movies.csv')
df_links = pd.read_csv('../../../../../Downloads/ml-25m/links.csv')
df_tags = pd.read_csv('../../../../../Downloads/ml-25m/tags.csv')
```

In [84]:
```
# Merge the datasets together to obtain one full dataframe.
merged_data = df_ratings.merge(df_movies,on='movieId', how='left')
```

In [85]:
```
# Create a new dataframe with userId, movieId, and rating.
df_user_recommended = merged_data[['userId', 'movieId', 'rating']].copy()

# Set the values to use for the csr matrix dimensions.
n_ratings = len(df_user_recommended)
n_movies = len(df_user_recommended['movieId'].unique())
n_users = len(df_user_recommended['userId'].unique())

# print the n ratings to ensure they are valid.
print(f"Total Movies: {n_movies} \nTotal Users: {n_users} \nTotal Ratings: {n_ratings}")
```

```
Total Movies: 59047
Total Users: 162541
Total Ratings: 25000095
```

In [86]:
```
# Map the indices to users and movie ids.
user_map = dict(zip(np.unique(df_user_recommended['userId']), list(range(n_users))))
movie_map = dict(zip(np.unique(df_user_recommended['movieId']), list(range(n_movies))))

user_i_map = dict(zip(list(range(n_users)), np.unique(df_user_recommended['userId'])))
movie_i_map = dict(zip(list(range(n_movies)), np.unique(df_user_recommended['movieId'])))

# Create indices for the csr matrix to be used next.
```

```
user_index = [user_map[i] for i in df_user_recommended['userId']]
movie_index = [movie_map[i] for i in df_user_recommended['movieId']]
```

In [87]:
```
# Create the csr_matrix for comparison
from scipy.sparse import csr_matrix

matrix = csr_matrix((df_user_recommended["rating"], (movie_index, user_index)), shape=(n
```

In [88]:
```
# Map the movies to the ids.
movie_names_mapped = dict(zip(merged_data['movieId'], merged_data['title']))
```

In [102…
```
'''
This function will use K nearest neighbors to determine the best matching movie from the
list will be returned along with a links.

@param: name - the name of the movie
@param: total_matches - the total number of movies to return
'''
from sklearn.neighbors import NearestNeighbors

def find_related_movies(name, total_matches):

    # Increment total matches since we'll be removing the one that matches the closest (
    total_matches += 1

    # Create a variable to hold our neighbors.
    neighbour_ids_with_distance = {}

    # Look up the movie the user entered with contains (less strict) and obtain the movi
    user_movie_id = next((k for k, v in movie_names_mapped.items() if v == name), None)

    # Prepare a vector for the KNN model.
    movie_index_mapped = movie_map[user_movie_id]
    movie_vector = matrix[movie_index_mapped]

    # Set the KNN model and fit it.
    knn = NearestNeighbors(algorithm = 'brute', metric='cosine')
    knn.fit(matrix)

    # reshape and determine distances for KNN values.
    movie_vector_reshaped = movie_vector.reshape(1, -1)
    distances, indices = knn.kneighbors(movie_vector_reshaped, n_neighbors=total_matches

    # Loop over the data and flatten the distances.
    for i in range(0,len(distances.flatten())):
        n = indices.flatten()[i]
        neighbour_id = movie_i_map[n]

        neighbour_ids_with_distance[movie_names_mapped[neighbour_id]] = distances.flatte

    # Remove the same movie from the list.
    neighbour_ids_with_distance.pop(movie_names_mapped[user_movie_id], None)

    # Sort the data by accuracy
    sorted_neighbours = sorted(neighbour_ids_with_distance.items(), key=lambda x: x[1],

    # Print the games and their related accuracy.
    count = 1
    movie_link = "https://www.themoviedb.org/movie/"

    print(f"Movies related to: {movie_names_mapped[user_movie_id]}\n")
    for movie, accuracy in sorted_neighbours:
        if count == total_matches:
            break
```

```
        else:
            # Set the movie link to pass with the data.
            next_movie_id = next((k for k, v in movie_names_mapped.items() if v == movie
            tmdb_id = df_links.loc[df_links['movieId'] == next_movie_id, 'tmdbId'].item(

            neighbour_movie_link = movie_link + str(int(tmdb_id))

            print(f"{movie}: {neighbour_movie_link}")
            count += 1
```

Test the Results

In [103…

```
# Test 1
movie_name = "Toy Story (1995)"
total_matches = 5

find_related_movies(movie_name, total_matches)
```

```
Movies related to: Toy Story (1995)

Star Wars: Episode IV - A New Hope (1977): https://www.themoviedb.org/movie/11
Toy Story 2 (1999): https://www.themoviedb.org/movie/863
Back to the Future (1985): https://www.themoviedb.org/movie/105
Forrest Gump (1994): https://www.themoviedb.org/movie/13
Jurassic Park (1993): https://www.themoviedb.org/movie/329
```

In [104…

```
# Test 2
movie_name = "Toy Story 2 (1999)"
total_matches = 10

find_related_movies(movie_name, total_matches)
```

```
Movies related to: Toy Story 2 (1999)

Toy Story (1995): https://www.themoviedb.org/movie/862
Bug's Life, A (1998): https://www.themoviedb.org/movie/9487
Monsters, Inc. (2001): https://www.themoviedb.org/movie/585
Shrek (2001): https://www.themoviedb.org/movie/808
Finding Nemo (2003): https://www.themoviedb.org/movie/12
Ghostbusters (a.k.a. Ghost Busters) (1984): https://www.themoviedb.org/movie/620
Chicken Run (2000): https://www.themoviedb.org/movie/7443
Men in Black (a.k.a. MIB) (1997): https://www.themoviedb.org/movie/607
Back to the Future (1985): https://www.themoviedb.org/movie/105
Sixth Sense, The (1999): https://www.themoviedb.org/movie/745
```

In [105…

```
# Test 3
movie_name = "Fight Club (1999)"
total_matches = 10

find_related_movies(movie_name, total_matches)
```

```
Movies related to: Fight Club (1999)

Matrix, The (1999): https://www.themoviedb.org/movie/603
Memento (2000): https://www.themoviedb.org/movie/77
American Beauty (1999): https://www.themoviedb.org/movie/14
Lord of the Rings: The Fellowship of the Ring, The (2001): https://www.themoviedb.org/mo
vie/120
Pulp Fiction (1994): https://www.themoviedb.org/movie/680
American History X (1998): https://www.themoviedb.org/movie/73
Lord of the Rings: The Return of the King, The (2003): https://www.themoviedb.org/movie/
122
Kill Bill: Vol. 1 (2003): https://www.themoviedb.org/movie/24
```

```
Lord of the Rings: The Two Towers, The (2002): https://www.themoviedb.org/movie/121
Dark Knight, The (2008): https://www.themoviedb.org/movie/155
```

Conclusion

Based on the data that were provided, I decided to use a K-Nearest Neighbors (KNN) model to generate predictions based on how close the ratings were with each respective movie title. Once the list of movies were collected, I looped through the results and created a link — using the links dataset — on the "tmdbId" column to allow the user to easily navigate to the movie suggestion for review. Based on the tests shown above with the movies "Toy Story", "Toy Story 2", and "Fight Club", we can easily see that the model has fairly decent accuracy in terms of recommendations. For instance, the test on "Toy STory" and "Toy Story 2" both reference one another, as they should, along with other movies that fall into either a children's movie theme or an action/adventure theme. Alternatively, the suggestions for "Fight Club" match a darker theme of movie with mentions such as "Kill Bill" and "American History X". Overall, the KNN recommendation model shows promise in generating accurate and relevant movie suggestions based on user input, while also providing links to the recommended movies for user perusal.