

The logo for MovieLens, featuring the word "movielens" in a white, lowercase, sans-serif font on an orange background.

Non-commercial, personalized movie recommendations.

[sign up now](#)

or [sign in](#)

Movie Recommender Systems

05.16.2019

[Dhaval Patel](#)

COMSC 415: Machine Learning

Professor Cates

Overview

In this assignment I will create a recommender system for movies and use it to make movie recommendations for yourself and your friends using the MovieLens database.

Goals

1. Create a Content-Based Recommendation system.
2. Create a Collaborative Filtering recommendation System.


Discussion/Methods

1. For the content based recommendation system, I decided to start by researching similarity measures and ways to vectorize the tags give. In the process however, I realized that the tags data wasn't complete and it would make for an incomplete recommendation system. My goal was to combine the tags and genres for every movie as to create keywords to use for the recommender.

During research, I read about the different similarity measures and decided to try out the ones I thought would be the most relevant for the task. I chose Cosine Similarity, Euclidean Distances and Manhattan Distances. The euclidean distances are good for dense matrices and because of this, I used a count-vectorizer and converted it to a dense matrix. I also used TF-IDF vectors for term-weighting but it did not give the best results for movies with higher amount of keywords. However, it performed better in movies without tags and just genres. This is most likely because of the inverse frequency weighting that TFIDF does. For movies with tags and genres, the count vectorizer performed slightly better.

The similarity measure I used for content-based recommendations was cosine similarity as it performed better overall. This is because magnitude is not a big factor in the case of keywords. I also decided to use the tfidf vectors instead of count vectors because cosine similarity performs better with vectors bound by $[0,1]$.

Movies recommended by my chosen method were very good according to me. As seen in the test runs for the content-based recommendations below, the movies



recommended for 'Batman Forever' were other superhero movies. The results therefore are reasonable and I would enjoy all the movies recommended to me here.

2. Collaborative Filtering requires similarity measures of user ratings. The ratings matrix created was used as a starting point and Jaccard Distances was tested as an additional similarity measure. The reason I tried Jaccard similarity in addition to the other measures is initially I thought that similar users would have common movies rated. However, I gave it more thought after tests were not giving me good results. The reason I think this similarity measure doesn't work is because it doesn't account for the ratings themselves and rather just considers what movies the users had rated in common.

The measure I decided to use was euclidean because euclidean distances take into account the magnitude and therefore the ratings themselves. For the purposes of the task of finding similar users, this method worked the best. After getting the most similar user, the sorted list of movies rated by the user was presented as recommendations.

Overall, the recommender systems performed well. Anecdotally, the movies recommended make sense knowing how the recommendations were gathered. For evaluation of the models I think looking at other more complex systems like the movielens website is a good way to judge if there are similarities between the system recommendations. Another good way to judge the recommender system is to use recall and precision values. In general, recall would be what the ratio of movies that a user likes were recommended and precision would be how many movies the user liked out of all the movies recommended. We would want to maximize both of these in order to get the best recommender system we can.

Results/Test Runs

I. Content-Based Recommendations Output

```

73
74 #Similarity/ Distance measures
75 cos_sim_count = cosine_similarity(X_count) # For count vectorizer
76
77 cos_sim_tfidf = linear_kernel(tf_idf,tf_idf) # Dot Product because of TFIDF vectors and faster processing
78
79 man_dist_count = manhattan_distances(X_count)
80
81 euc_dist_count = euclidean_distances(X_dense)
82
83 euc_dist_tfidf = euclidean_distances(tf_idf)
84
85
86 #reverse Lookup of title and movie indices
87 df_movie_indices = pd.Series(df_movies.index, index=df_movies['title']).drop_duplicates()
88
89 def recommend_content(title, similarity, matrix): # Change cosine similarity matrices here (TFIDF/Count)
90
91     index = df_movie_indices[title]
92
93
94     # Get the pairwise similarity scores of all movies with that movie
95     sim_scores = list(enumerate(matrix[index]))
96
97     # Sort the movies based on the similarity scores
98     if (similarity == 'euc') or (similarity == 'man'):
99         sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=False) # Lower distance = higher similarity
100     else:
101         sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True) # cosine
102
103
104     # Get the scores of the 10 most similar movies
105     sim_scores = sim_scores[1:11]
106
107
108     # Get the movie indices
109     movie_indices = [i[0] for i in sim_scores]
110
111
112     # Return the top 10 most similar movies
113     return df_movies['title'].iloc[movie_indices]
114
115
116 print ("Content Based Recommender: \n\n", recommend_content('Batman Forever (1995)', 'cos', cos_sim_tfidf))
117 print()
118
119 #Part 4 user recommendation
120
121
122 #Similarity/ Distances
123 cosine_sim_ratings = cosine_similarity(df_ratings_matrix)
124 euc_dist_ratings = euclidean_distances(df_ratings_matrix)
125 man_dist_ratings = manhattan_distances(df_ratings_matrix)
126
127 df_ratings_matrix_new = df_ratings_matrix.values # ndarray matrix for easier iteration
128
129 #Function for pairwise iaccard

```

Name	Type	Size	
X_dense	int64	(9742, 1748)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
cos_sim_count	float64	(9742, 9742)	[[1. 0.3380617 0.]

Help Variable explorer File explorer

IPython console

Console 1/A

```

In [11]: print ("Content Based Recommender: \n\n", recommend_content('Jumanji', 'cos', cos_sim_tfidf))
Content Based Recommender:

53         Indian in the Cupboard, The (1995)
109         NeverEnding Story III, The (1994)
767         Escape to Witch Mountain (1975)
1514         Darby O'Gill and the Little People (1959)
1556         Return to Oz (1985)
1617         NeverEnding Story, The (1984)
1618         NeverEnding Story II: The Next Chapter, The (1...
1799         Santa Claus: The Movie (1985)
6389         Bridge to Terabithia (2007)
6629         Golden Compass, The (2007)
Name: title, dtype: object

In [12]: print ("Content Based Recommender: \n\n", recommend_content('Batman Returns (1992)', 'cos', cos_sim_tfidf))
Content Based Recommender:

1060         Batman Returns (1992)
99         Rumble in the Bronx (Hont faan kui) (1995)
509         Batman (1989)
2044         Mystery Men (1999)
2897         Supergirl (1984)
3511         It's a Mad, Mad, Mad, Mad World (1963)
4071         I Spy (2002)
4726         Nothing to Lose (1997)
6274         Crime Busters (1977)
1986         Superman (1978)
Name: title, dtype: object

In [13]: print ("Content Based Recommender: \n\n", recommend_content('Batman Returns (1992)', 'cos', cos_sim_tfidf))
Content Based Recommender:

1060         Batman Returns (1992)
509         Batman (1989)
2897         Supergirl (1984)
2044         Mystery Men (1999)
1986         Superman (1978)
1988         Superman III (1983)
1987         Superman II (1980)
3819         Spider-Man (2002)
2836         X-Men (2000)
7693         Avengers, The (2012)
Name: title, dtype: object

```

II. Collaborative Filtering Recommendations Output

The screenshot displays a Jupyter Notebook interface with a code editor on the left and a console/output area on the right. The code in the notebook implements a collaborative filtering recommendation system. It starts by creating a matrix of user ratings, then finds similar users for a given input user. Finally, it recommends movies based on the ratings of these similar users, excluding movies the input user has already rated.

Code Snippet (Left Panel):

```

174 #Part 4 movie recommendation (collaborative)
175
176 df_ratings_new = df_ratings.as_matrix(columns = ['userId', 'movieId', 'rating'])
177 df_ratings_new = (df_ratings.sort_values('userId'))
178 df_ratings_new = df_ratings_new.as_matrix(columns = ['userId', 'movieId', 'rating'])
179
180
181 #Create Each Users ratings List
182 users_list = []
183 for i in range(1,611):
184     list1 = []
185     for j in range(0, len(df_ratings_new)):
186         if df_ratings_new[j][0] == i:
187             list1.append(df_ratings_new[j])
188         else:
189             break
190 df_ratings_new = df_ratings_new[j:]
191 users_list.append(list1)
192
193
194 def recommend_movies_collab(input_user, similarity, matrix):
195
196     sim_user = int(recommend_users(input_user, similarity, matrix).loc[0]['userId'])
197
198     #Get sorted lists for users (highest rating to lowest)
199     input_user_list = sorted(users_list[input_user - 1], key=itemgetter(2), reverse = True)
200     sim_user_list = sorted(users_list[sim_user], key=itemgetter(2), reverse = True)
201
202     #Exclude common movies between users
203     common_list = []
204     full_list = []
205     for i in input_user_list:
206         for j in sim_user_list:
207             if(int(i[1]) == int(j[1])):
208                 common_list.append(int(j[1]))
209             full_list.append(j[1])
210
211     common_list = set(common_list)
212     full_list = set(full_list)
213     recommendation = list(full_list.difference(common_list))
214
215     #for Reverse Lookup of movieIds and movie indices
216     df_movieId_indices = pd.Series(df_movies.index, index=df_movies['movieId']).drop_duplicates()
217
218     movieId_indices = [int(i) for i in recommendation]
219     movie_indices = []
220
221     #Get movie indices
222     for i in movieId_indices:
223         movie_indices.append(df_movieId_indices[i])
224
225     return df_movies['title'].iloc[movie_indices]
226
227
228
229 print("Movies recommended by Collaborative approach: \n\n", recommend_movies_collab(50, 'euc', euc_dist_ratings) )
230 print()
231
232

```

Output (Right Panel):

The output shows the results of the recommendation process. It includes a table of similar users and a list of recommended movies.

Name	Type	Size	Value
i	int	1	610

IPython console

Console 1/A

10 Similar users (most to least):

userId	Similarity/Distance
0	440 51.577611
1	291 51.802510
2	547 51.966335
3	229 51.968741
4	431 52.144031
5	327 52.153619
6	243 52.235046
7	394 52.268537
8	429 52.299618
9	476 52.321124

Movies recommended by Collaborative approach:

- 3638 Lord of the Rings: The Fellowship of the Ring,...
- 224 Star Wars: Episode IV - A New Hope (1977)
- 7018 Star Trek (2009)
- 8683 Star Wars: Episode VII - The Force Awakens (2015)
- 7768 Dark Knight Rises, The (2012)
- 9 GoldenEye (1995)
- 7090 District 9 (2009)
- 461 Schindler's List (1993)
- 2038 Ghostbusters (a.k.a. Ghost Busters) (1984)
- 5335 Shaun of the Dead (2004)
- 898 Star Wars: Episode V - The Empire Strikes Back...
- 43 Seven (a.k.a. Se7en) (1995)
- 3256 Dr. Goldfoot and the Bikini Machine (1965)
- 2440 Wayne's World (1992)
- 2441 Wayne's World 2 (1993)
- 6422 Hot Fuzz (2007)
- 7010 Inglourious Basterds (2009)
- 828 Reservoir Dogs (1992)
- 1979 Star Wars: Episode I - The Phantom Menace (1999)
- 507 Terminator 2: Judgment Day (1991)
- 176 Waterworld (1995)
- 5350 Team America: World Police (2004)
- 940 Dead Alive (Braindead) (1992)
- 3363 Bill & Ted's Excellent Adventure (1989)
- 4927 Dawn of the Dead (1978)
- 92 Happy Gilmore (1996)
- 3903 Austin Powers in Goldmember (2002)
- 7633 Your Highness (2011)
- 1154 Austin Powers: International Man of Mystery (1...
- 7154 Zombieland (2009)
- 4800 Lord of the Rings: The Return of the King, The...
- 3629 Bill & Ted's Bogus Journey (1991)

IPython console History log

Sources

Andersen, Kagemand AndersenKagemand. "Compute Jaccard Distances on Sparse Matrix." *Stack Overflow*,
stackoverflow.com/questions/32805916/compute-jaccard-distances-on-sparse-matrix.

"Building A Recommender System on User-User Collaborative Filtering (MovieLens Dataset)." *Code for Thought*, 29 Dec. 2016,
acodeforthought.wordpress.com/2016/12/29/building-a-recommender-system-on-user-user-collaborative-filtering-movielens-dataset/.

Jain, Aarshay. "Quick Guide to Build a Recommendation Engine in Python." *Analytics Vidhya*, 22 Feb. 2019,
www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/.

"Recommender Systems in Python Tutorial." *DataCamp Community*,
www.datacamp.com/community/tutorials/recommender-systems-python.

"Recommender Systems through Collaborative Filtering." *Data Science Blog by Domino*,
blog.dominodatalab.com/recommender-systems-collaborative-filtering/.