Movie Recommendation System

Hatim Alhazmi, Pavan Patel
Department Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07307, USA
Email: halhazm1@stevens.edu, ppate105@stevens.edu

***Abstract*—Recommendation systems could suggest and help users in their choices. In this paper, we plan to build a movie recommendation system. The most used approaches in the area of movie recommendation systems are Content based filtering and collaborative filtering. The data of movies and users can be translated in order of making new movie recommendations. We will use TensorFlow to build a movie recommendation system by applying Natural Language Processing and Neural Networks.**

***Index Terms*—Movie recommendation, TensorFlow, NLP, Neural Networks, Content based filtering, Collaborative**

I. Introduction

Nowadays, recommendations are playing an important role in our life day by day. It helps us to make some decisions and make a course of actions by using some external knowledge, for instance when you plan to visit a doctor or go to the movie theater. This knowledge can be derived from social processes. Sometimes we rely on people's judgement when we are buying a CD, who shares similar tastes in music. At other times, available information about the artifact itself plays an important role in the judgments. Many factors can influence people making their choices, and the person would like to model as many of the possible factors in a recommendation system.

Speaking of movie platforms these days, there are so many collections of websites movies for example, Netflix, Amazon, Hulu, etc. that amount of platforms make the people confused about choosing what to watch. This problem led us to a movie recommendation system and how it comes in. A movie recommendation system can work even if you are a new user or not . For a new user who has not watched any movie before on this platform, he will be recommended by the top movies. For fairly new users, it takes the most recent movie that the user has watched earlier and recommends him movies similar to his most recent experiences. This way the movie recommendation system presents both new and existing users. Our goal to build a similar movie recommendation system that implements both content based filtering and collaborative filtering. In order to do this we will use Natural Language Processing and Neural Networks.

II. Related Work

In research, there are some general approaches and solutions related to this problem. Some of the solutions include algorithms like Cosine Similarity, Euclidean Similarity, K nearest neighbors, Single Value Decomposition. These existing solutions cannot process all the users and also solve only one of the two either content based or collaborative filtering. Based on that we are planning to build a movie recommendation system that can deal with any kind of user.

III. Problem Description

Few years ago, you would have to go to the Blockbuster store and buy DVDs of your favorite movies but in this day and age of the internet it's rather the opposite, nowadays the internet allows people to access abundant resources online. There are enormous collections of movies on websites like Netflix, Amazon, Hulu, etc. People are having a hard time choosing what to watch. This is where a movie recommendation system comes in. The system works for both the new users and existing users. Let's suppose there is a new subscriber to the service, obviously the subscriber has not watched a single movie, it recommends him/her the all time top movies. For fairly new subscribers, it takes the most recent movie the user has watched and recommends him/her movies similar to his most recent movie. For an existing subscriber, their recommendation system recommends the subscriber new movies that other users, who have similar taste. This way the movie recommendation system addresses both new and existing users. Our aim is to build a similar hybrid movie recommendation system that implements both content based filtering and collaborative filtering. We will do this using Natural Language Processing and Neural Networks.

IV. Dataset Description

We are using MI-20M: This Dataset ml-20m describes 5 star rating and free -text tagging activity from Movielens, a movie recommendation service. Contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. The data are contained in six files - genome - scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv.his dataset shown in Fig. 3 is only good enough for collaborative Filtering method but is not good enough for content-based recommendation system.

So, we used ImdbPY to get additional information like the director's name, plot and cast. The ImdbPY is a python package for extracting and managing the data of the IMDB database.As it consists of movies, directors, cast, plot etc, this

additional information can be helpful for suggesting movies to new users or users who have rated less movies. Figure-1 shows the ratings.csv file which contains the movie-id, user-id and the respective ratings. Figure-2 shows the movies.csv file which contains movie-id, the movie title and its respective genre.

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Fig. 1. Movielens dataset.

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Fig. 2. Movies.csv and ratings.csv merged dataset.

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483

Fig. 3. Ratings and Movie merged Dataset.

V. Our Solution

Our solution to implement content based filtering and collaborative filtering so that we can address both new users and existing users. We will implement content-based filtering using Natural Language Processing that takes in data such as director, cast and plot of the movie for new users. We will implement collaborative filtering using Neural Networks that takes in data such as user-Id, movie-Id and rating for existing users. This will result in our ultimate goal.

A. Exploratory Data Analysis

One of the approaches is exploratory Data Analysis which is to analyze data sets in order to summarize their main characteristics, with visual methods. Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, spot anomalies, test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

B. Machine learning algorithm used

As mentioned earlier we will implement both content based filtering and collaborative filtering so that we can address both new users and existing users. In content-based filtering we intend to implement Natural language processing and Term Frequency (TF) and Inverse Document Frequency (IDF) while in collaborative filtering we intend to implement K nearest neighbors and Neural Networks.

a. Natural Language Processing (NLP)

An important part of machine learning which aids computers to analyze, elucidate and manipulate human language is Natural Language processing (NLP). Natural Language Processing includes various approaches for elucidating human language, from statistical and machine learning systems to standards- based and analytical approaches. The gap between computer and human linguistics is reduced by NLP as it being an area of computational linguistics and computer science, it is used by the computer. We need a broad wide-ranging method because the voice and test-based data varies broadly, as do the practical utilization. It can be used for content categorization, sentiment analysis, machine translation, document summarization or contextual extraction. We are using NLP to find similarities between the plots of different movies as NLP uses parts of speech to identify semantic relationships.

We have selected Natural Language Processing as we believe it will provide more accurate and personalized results to the user. While implementing Natural Language Processing on the final big dataset that we have gathered, we took the fields several fields like title, Director, Cast, genre, Plot and load it into a dataframe as shown in Fig 4.

	title	Director	Cast	genres	Plot
0	Toy Story (1995)	John Lasseter	Tom Hanks	Adventure|Animation|Children|Comedy|Fantasy	A cowboy doll is profoundly threatened and jea...
1	Jumanji (1995)	Joe Johnston	Robin Williams	Adventure|Children|Fantasy	When two kids find and play a magical board ga...
2	Grumpier Old Men (1995)	Howard Deutch	Walter Matthau	Comedy|Romance	John and Max resolve to save their beloved bai...
3	Waiting to Exhale (1995)	Forest Whitaker	Whitney Houston	Comedy|Drama|Romance	Based on Terry McMillan's novel, this film fol...
4	Father of the Bride Part II (1995)	Charles Shyer	Steve Martin	Comedy	George Banks must deal not only with the pregn...

Fig. 4. Original Data.

After this we proceed to simplify out dataset as our dataset is too complex before we feed it to the Natural Language Processing algorithm so as to get more precise results. First, we simplified the genre into maximum 2 genre that best define

a movie. Further we also combine the first name and last name of every single person into a unique single name for directors and cast names. This will make sure that there are no duplicates. We also converted everything into lower case. (refer Fig.5)

	title	Director	Cast	genres	Plot
0	Toy Story (1995)	johnlasseter	tomhanks	[adventure, animation]	a cowboy doll is profoundly threatened and jea...
1	Jumanji (1995)	joejohnston	robinwilliams	[adventure, children]	when two kids find and play a magical board ga...
2	Grumpier Old Men (1995)	howarddeutch	waltermatthau	[comedy, romance]	john and max resolve to save their beloved bai...
3	Waiting to Exhale (1995)	forestwhitaker	whitneyhouston	[comedy, drama]	based on terry mcmillan's novel, this film fol...
4	Father of the Bride Part II (1995)	charlesshyer	stevemartin	[comedy]	george banks must deal not only with the pregn...

Fig. 5. Data Simplifying.

Now as we already have the simplified data ready, we move on with the actual Natural Language Processing Algorithm. First, we take plot form the simplified data and run it through the algorithm. We have implemented the Natural Language Algorithm using the 'Rake-nltk' library. This will return us a list of key words that it thinks are most important words in movie plot as shown in Fig.6.

title				
Toy Story (1995)	johnlasseter	tomhanks	[adventure, animation]	[profoundly, threatened, boy, jealous, room, t...
Jumanji (1995)	joejohnston	robinwilliams	[adventure, children]	[play, game, finishing, stopped, release, man,...
Grumpier Old Men (1995)	howarddeutch	waltermatthau	[comedy, romance]	[new, female, owner, catches, max, italian, re...
Waiting to Exhale (1995)	forestwhitaker	whitneyhouston	[comedy, drama]	[novel, terry, mcmillan, relationships, film, ...
Father of the Bride Part II (1995)	charlesshyer	stevemartin	[comedy]	[unexpected, pregnancy, george, banks, must, d...

Fig. 6. Fetching important words.

Now, as we have a list of unique words that best describe the plot of a movie, we then proceed with putting all of our names and list of words into a 'bag of words'. We thus assigned a bag of words which contains everything from our data frame to every single movie. Refer Fig.7. The idea we are trying to implement is to compare the bag of words of each movie and decide which movies are similar to the selected movie. As seen in Figure.8 that if we make recommendations for the movie Batman we see other Batman movies as suggestions and as well as other very similar movies.

b. Term Frequency (TF) and Inverse Document Frequency (IDF).

TF-IDF is a product of "Term Frequency (TF)" and "Inverse Document Frequency (IDF)". TF shows how many times a subject term appears in the document. IDF is the inverse of the document frequency for the documents which contain the subject term. In this practice, TF means how frequently a tag is applied for a movie and IDF indicates how rarely a tag is

title	
Toy Story (1995)	johnlasseter tomhanks adventure animation prof...
Jumanji (1995)	joejohnston robinwilliams adventure children p...
Grumpier Old Men (1995)	howarddeutch waltermatthau comedy romance new ...
Waiting to Exhale (1995)	forestwhitaker whitneyhouston comedy drama nov...
Father of the Bride Part II (1995)	charlesshyer stevemartin comedy unexpected pre...

Fig. 7. Bag Words.

```
['Batman Beyond: Return of the Joker (2000)',
 'Justice League: Doom (2012) ',
 'Batman: Assault on Arkham (2014)',
 'Batman Begins (2005)',
 'Batman/Superman Movie, The (1998)',
 'Batman (1989)',
 'Dead Presidents (1995)',
 'History of Violence, A (2005)',
 'Helter Skelter (2004)',
 'The Godfather Trilogy: 1972-1990 (1992)']
```

Fig. 8. Making Movie Recommendations.

applied. TF-IDF is calculated using the equation as shown in Fig.9.

$$\mathbf{TFIDF}_{i,j} = TF_{i,j} * \log\left(\frac{N}{DF_i}\right)$$

Fig. 9. TF-IDF Equation.

TF-IDF lets us score the terms more reliably. For example, let's assume that we have a set of terms "the ultimate action" to describe a movie. The term "the" would occur more frequently than "action" in the dataset, however, it is less important in evaluating the similarities than "action". TF-IDF helps to penalize these terms so that the similarity computation can be less affected.hat means that words such as 'is', 'are', 'by' or 'a' which are likely to show up in every movie description but aren't useful for our user-recommendation, will be weighed less than words that are more unique to the content that we are recommending. A vector-encoded document will look like as shown in Fig.10. when encoded.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1.000000	0.408613	0.171842	0.141888	0.208914	0.244326	0.235981	0.257648	0.084780	0.252850	0.240785	0.069502	0.376399	0.176336	0.120842

Fig. 10. vector-encoded document.

Each element in the vector represents a TF-IDF weight associated with a term in a document. For recommending movies we again used the Cosine Similarity explained earlier.For example, the movies Toy Story and Monsters, Inc have a cosine similarity of 0.74.. In contrast, the cosine similarity between the movies Toy Story and Terminator 2 is 0.28 - as expected much lower (as shown in Fig.11).

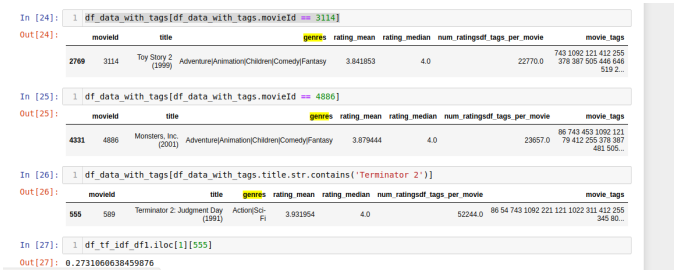

Fig. 11. vector-encoded document.

We can now recommend movies based on the movies that a user has already watched or rated using the cosine similarity. We would recommend movies with the largest similarity to the ones already highly rated by the user. So for recommendation the idea we implemented was instead of recommending movies based on specific movies that a user has already watched, we could also attempt to build profiles of the users preferences as shown in Fig.12.

	title	rating
0	Jumanji (1995)	3.5
1	City of Lost Children, The (Cité des enfants p...	3.5
2	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	3.5
3	Seven (a.k.a. Se7en) (1995)	3.5
4	Usual Suspects, The (1995)	3.5
...	...	...
169	Freaks (1932)	5.0
170	Spider-Man 2 (2004)	4.5
171	Slaughterhouse-Five (1972)	3.5
172	Incredibles, The (2004)	4.0
173	Constantine (2005)	4.0

174 rows × 2 columns

Fig. 12. Building Profile for user 1.

This will allow us to gain an aggregate view of the users' preferences and then recommend content based on their behaviour over time without skewing the recommendations by outliers. So if we take user 1 from the dataset. This user has rated the following movies from 1: dislike to 5: like.

For making the profile of a user we took the less principled approach. We took the weighted mean of the user's ratings and the TF-IDF vector representations of the respective movies. This simple weighted mean will then constitute the user's preference profile.Then all we did was, take the cosine similarity between the user profile vectors and content vectors to find their similarity. Now we can recommend the most similar items.

c. Neural Networks(NN).

We use Neural Networks For collaborative filtering. The idea behind using NN because it gave us the least error as compared to other algorithms. We implemented Neural Networks using Tensorflow. The data set that we have prepared as title, userId, movieId, rating as shown in Fig.13.

	title	userId	movieId	rating
0	Toy Story (1995)	1	1	4.0
1	Toy Story (1995)	5	1	4.0
2	Toy Story (1995)	7	1	4.5
3	Toy Story (1995)	15	1	2.5
4	Toy Story (1995)	17	1	4.5

Fig. 13. Neural Network Weights and Biases.

In our implementation we compare the tastes of different users based on the movies that they already have watched and rated before and recommend to them movies that are almost similar to their previous choices. The weights and biases are assigned values randomly as shown in Fig.14.

```
weights = {
    'encoder_w1': tf.Variable(tf.random_normal([number_input, number_hidden_1], dtype=tf.float64)),
    'encoder_w2': tf.Variable(tf.random_normal([number_hidden_1, number_hidden_2], dtype=tf.float64)),
    'decoder_w1': tf.Variable(tf.random_normal([number_hidden_2, number_hidden_1], dtype=tf.float64)),
    'decoder_w2': tf.Variable(tf.random_normal([number_hidden_1, number_input], dtype=tf.float64)),
}

biases = {
    'encoder_b1': tf.Variable(tf.random_normal([number_hidden_1], dtype=tf.float64)),
    'encoder_b2': tf.Variable(tf.random_normal([number_hidden_2], dtype=tf.float64)),
    'decoder_b1': tf.Variable(tf.random_normal([number_hidden_1], dtype=tf.float64)),
    'decoder_b2': tf.Variable(tf.random_normal([number_input], dtype=tf.float64)),
```

Fig. 14. Neural Network Weights and Biases.

For optimizing our neural network, we measure the losses as mean squared error. We then use the RMSPropOptimizer from tensorflow to minimize the losses (as shown in Fig.15.

```
#calculate Mean squared error between orginal data and predect data
loss = tf.losses.mean_squared_error(y_true, y_pred)
optimizer = tf.train.RMSPropOptimizer(0.03).minimize(loss)
evaluate_x = tf.placeholder(tf.int32, )# evaluate
evaluate_y = tf.placeholder(tf.int32, )
pre, pre_opration = tf.metrics.precision(labels=evaluate_x, predictions=evaluate_y)
```

Fig. 15. Optimization.

Speaking about making the recommendations, first see the movies that have been rated by a random user, say userId 95as shown in Fig.16.

So, based on these ratings, we recommend the user new movies to watch as shown in Fig.17.

	userId	title	rating
214051	95	Forrest Gump (1994)	0.666415
214008	95	Fight Club (1999)	0.628395
214894	95	Pulp Fiction (1994)	0.601369
215110	95	Silence of the Lambs, The (1991)	0.527130
215088	95	Shawshank Redemption, The (1994)	0.518397
214505	95	Lord of the Rings: The Fellowship of the Ring,...	0.484920
215037	95	Schindler's List (1993)	0.478865
215125	95	Sixth Sense, The (1999)	0.472321
214117	95	Godfather, The (1972)	0.447260
214506	95	Lord of the Rings: The Return of the King, The...	0.426658

Fig. 16. Neural Network Weights and Biases.

	title	userId	movieId	rating
23166	Reservoir Dogs (1992)	95	1089	5.0
26646	Godfather: Part II, The (1974)	95	1221	5.0
50735	Drugstore Cowboy (1989)	95	3019	5.0
24290	Delicatessen (1991)	95	1175	5.0
24679	Star Wars: Episode V - The Empire Strikes Back...	95	1196	5.0
25030	Raiders of the Lost Ark (Indiana Jones and the...	95	1198	5.0
38615	Saving Private Ryan (1998)	95	2028	5.0
25577	Clockwork Orange, A (1971)	95	1206	5.0
25759	Apocalypse Now (1979)	95	1208	5.0
64651	Amelie (Fabuleux destin d'Amélie Poulain, Le) ...	95	4973	5.0

Fig. 17. Neural Network Weights and Biases.

As you can see, the movies are related and these recommendations are fairly accurate for our user 95.

d.n-nearest neighbors

For n nearest neighbors we used a score function based on ratings and users as shown in Fig.18.

$$s(u,i) = \bar{r}_u + \frac{\sum_{v \in V}(r_{vi} - \bar{r}_v) * w_{uv}}{\sum_{v \in V} w_{uv}}$$

score function

Fig. 18. n nearest neighbors score function.

Where 's' is the predicted score, 'u' is the user, 'i' is the item, 'r' is the rating given by the user and 'w' is the weight.In this case our score is equal to the sum of the ratings that each user gave to that item subtracting the average rating of that user multiplied with some weight which is of how much this user is similar or supposed to contribute to the predictions of other user. This is the weight between user u and v. The score ranges between 0 to 1 where 0 is low and 1 is high.The problem is with the types of users we are handling. Some people may be positive and optimistic users where they will rate the movie they liked as 4 out of 5 but some other users who are less optimistic or have some high standards may rate there favorite movie as 2 out of 5. Here their 2 is our 4. The tweaks we made to increase the efficiency of this algorithm were to normalize user's ratings. Our way to do that was to compute s(u,i) i.e score as the average rating that user gives to each item plus some deviation and the deviation is going to be how much this item is better or worse than average.

We have again used the cosine similarity to calculate the weight given in the above formula. We have also used the notion of neighborhood which is explained later on. So, after normalizing the data is as shown in Fig.19.

```
df_movies = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Machine Learning Data/movies.csv')
df_ratings = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Machine Learning Data/ratings.csv')
df_tags = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Machine Learning Data/tags.csv')

Mean = df_ratings.groupby(by="userId",as_index=False)['rating'].mean()
df_ating_avg = pd.merge(df_ratings,Mean,on='userId')
df_rating_avg['adg_rating']=df_rating_avg['rating_x']-df_rating_avg['rating_y']
df_rating_avg.head()
```

	userId	movieId	rating_x	timestamp	rating_y	adg_rating
0	12882	1	4.0	1147195252	4.061321	-0.061321
1	12882	32	3.5	1147195307	4.061321	-0.561321
2	12882	47	5.0	1147195343	4.061321	0.938679
3	12882	50	5.0	1147185499	4.061321	0.938679
4	12882	110	4.5	1147195239	4.061321	0.438679

Fig. 19. Normalization.

We used this normalized rating data of a user to calculate the final score for the user later. Also before applying the cosine similarity we needed to clean up the data as it had lots of NaN value since every user has not seen all the movies. This is shown in Fig.20.

```
final = pd.pivot_table(df_rating_avg,values='adg_rating',index='userId',columns='movieId')
final.head()
```

movieId	1	2	3	4	5	6	7	9	10	11	...	106487	106489	106782	106920	10937
userId																
316	-0.829457	NaN	NaN	NaN	NaN	NaN	-1.329457	NaN	-0.829457	NaN	...	NaN	NaN	NaN	NaN	Na
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	Na
359	1.314526	NaN	NaN	NaN	NaN	1.314526	NaN	NaN	0.314526	0.314526	...	NaN	NaN	NaN	NaN	Na
370	0.705596	0.205596	NaN	NaN	NaN	1.205596	NaN	NaN	NaN	NaN	...	-1.294404	-0.794404	0.705596	0.205596	Na
910	1.101920	0.101920	-0.39808	NaN	-0.39808	-0.398080	NaN	NaN	NaN	0.101920	...	NaN	NaN	-0.398080	NaN	Na

5 rows × 2500 columns

Fig. 20. NaN value.

So in order to remove the NaN we used 2 methods: 1- Use the user average over the row as shown in Fig.21. 2- User the movie average over the column as shown in Fig.22.

Next step is to calculate the similarity between the users as shown in Fig.23.

Then we checked that if calculated correctly or not as shown in Fig.24.

From the above image we can see that the similarity we generated is true since both the given users (370,86309) have almost the same ratings and liking's. So from above we can see that we have calculated the similarities for all the users.But this recommendation system works with the huge data and hence it becomes very important to maintain and capture only

```
# Replacing NaN by Movie Average
final_movie = final.fillna(final.mean(axis=0))

# Replacing NaN by user Average
final_user = final.apply(lambda row: row.fillna(row.mean()), axis=1)
```

```
final_movie.head()
```

movieId	1	2	3	4	5	6	7	9	10	11	...	106487	106489	106782	1
userId															
316	-0.829457	-0.436518	-0.468109	-0.770223	-0.615331	0.320415	-1.329457	-0.690175	-0.829457	-0.094277	...	0.105075	0.006629	0.262314	0.1
320	0.200220	-0.436518	-0.468109	-0.770223	-0.615331	0.320415	-0.203889	-0.690175	-0.150642	-0.094277	...	0.105075	0.006629	0.262314	0.1
359	1.314526	-0.436518	-0.468109	-0.770223	-0.615331	1.314526	-0.203889	-0.690175	0.314526	0.314526	...	0.105075	0.006629	0.262314	0.1
370	0.705596	0.205596	-0.468109	-0.770223	-0.615331	1.205596	-0.203889	-0.690175	-0.150642	-0.094277	...	-1.294404	-0.794404	0.705596	0.2
910	1.101920	0.101920	-0.398080	-0.770223	-0.398080	-0.398080	-0.203889	-0.690175	-0.150642	0.101920	...	0.105075	0.006629	-0.398080	0.1

5 rows × 2500 columns

Fig. 21. movie average over the row.

```
final = pd.pivot_table(df_rating_avg,values='adg_rating',index='userId',columns='movieId')
final.head()
```

movieId	1	2	3	4	5	6	7	9	10	11	...	106487	106489	106782	106920	10937
userId																
316	-0.829457	NaN	NaN	NaN	NaN	NaN	-1.329457	NaN	-0.829457	NaN	...	NaN	NaN	NaN	NaN	Na
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	Na
359	1.314526	NaN	NaN	NaN	NaN	1.314526	NaN	NaN	0.314526	0.314526	...	NaN	NaN	NaN	NaN	Na
370	0.705596	0.205596	NaN	NaN	NaN	1.205596	NaN	NaN	NaN	NaN	...	-1.294404	-0.794404	0.705596	0.205596	Na
910	1.101920	0.101920	-0.39808	NaN	-0.39808	-0.398080	NaN	NaN	NaN	0.101920	...	NaN	NaN	-0.398080	NaN	Na

5 rows × 2500 columns

Fig. 22. movie average over the column.

```
# user similarity on replacing NAN by user avg
b = cosine_similarity(final_user)
np.fill_diagonal(b, 0 )
similarity_with_user = pd.DataFrame(b,index=final_user.index)
similarity_with_user.columns=final_user.index
similarity_with_user.head()
```

userId	316	320	359	370	910	975	1015	1387	1447	1588	...	137118	137209	137227	13744
userId															
316	0.000000	0.060063	0.072075	0.043266	0.039305	0.045616	0.035341	0.038068	-1.248514e-02	0.050183	...	0.052632	0.104864	0.011358	0.02967
320	0.060063	0.000000	0.063054	0.027315	0.006811	0.075620	0.011910	0.042509	-1.833485e-31	0.067389	...	0.115325	0.065130	0.071996	0.09755
359	0.072075	0.063054	0.000000	0.135836	0.076131	0.036757	0.046418	0.066544	4.287659e-02	0.109726	...	0.120191	0.020672	0.032166	0.03959
370	0.043266	0.027315	0.135836	0.000000	0.108404	0.071655	0.070893	-0.003139	5.223516e-02	0.090241	...	0.091218	0.049594	0.004344	0.04069
910	0.039305	0.006811	0.076131	0.108404	0.000000	0.021814	0.027339	-0.032211	-6.301121e-03	-0.007491	...	0.039464	-0.017620	0.020058	-0.00458

5 rows × 862 columns

Fig. 23. the similarity between the users.

```
a = get_user_similar_movies(370,86309)
a = a.loc[ : , ['rating_x_x','rating_x_y','title']]
a.head()
```

	rating_x_x	rating_x_y	title
0	5.0	5.0	Matrix, The (1999)
1	5.0	4.5	Lord of the Rings: The Fellowship of the Ring,...
2	5.0	4.0	Lord of the Rings: The Two Towers, The (2002)
3	4.5	4.0	Lord of the Rings: The Return of the King, The...
4	1.5	1.0	Serenity (2005)

Fig. 24. the similarity between the users.

the important and necessary highlights from the data. Hence to overcome this we generate a notion of neighborhood. This includes only the set of (K) similar users for a particular user.We have taken the value of k as 30. So we would have 30 nearest neighbors for all the users.We have used a custom function find n neighbours which takes the similarity matrix and the value of n as input and returns the nearest n neighbors for all the users.Due to this we have reduced the number of unnecessary computations as shown in Fig.25.

```
# top 30 neighbours for each user
sim_user_30_m = find_n_neighbours(similarity_with_movie,30)
sim_user_30_m.head()
```

	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24	top25	top26	top27
userId																		
316	138176	100240	96936	51460	88932	1447	104732	125012	5268	121403	...	121987	72633	21401	114335	22338	118304	124981
320	138176	96936	121403	1447	51460	125012	88932	42944	5268	104529	...	121987	102549	118304	86309	94333	124981	93203
359	138176	1447	5268	96936	100240	21401	88932	13927	104732	72633	...	12930	121987	114335	125012	51460	118304	57474
370	86309	44194	138176	24802	129869	96936	1447	104529	94333	88932	...	124981	27142	102549	120308	54643	42944	80585
910	96936	107991	138176	27142	51460	125012	88932	100240	72633	129869	...	36624	51255	94333	42944	121403	80585	61755

5 rows × 30 columns

Fig. 25. unnecessary computations.

Now we will try to predict the score for the movie the given user has not seen as shown in Fig.26.

```
score = User_item_score(320,7371)
print("score (u,i) is",score)
```

```
score (u,i) is 4.255766437391595
```

Fig. 26. Score prediction test.

So our system predicted the score to be 4.25 which is really good. We think user (370) could like the movie with id (7371). We already generated the score for one item. Similarly we can generate the score for the other items with the same user. We are just interested in calculating the scores for the items that their neighbor users have seen.User item score1 is our custom function which uses our above discussion to calculate predictions.

Overall, we recommend the movies using content based filtering to new users, and using collaborative filtering to existing users.

VI. Results

A. Observations and Comparison: All the algorithms that we considered for our recommendation system, Natural Language Processing and Neural Networks came with the least Mean Square Error. The computational cost may be higher than the rest of the algorithms but we are more focused on achieving accurate recommendations. See Table 1.

B. Advantages and Disadvantages:

Cosine Similarity:It is a simple to implement algorithm but has one drawback in our application in movie recommendation system. The drawback is the difference in rating scale between different users are not taken into account.

TABLE I
MSE AND COMPUTATIONAL COST

Algorithms	Computational Cost	MSE
NLP	HIGH	8211
TF-IDF	LOW	0.8431
NN	Depends on data	0.8099
N-nearest neighbors	LOW	0.8182

TD-IDF: A drawback of the weighted mean approach used in TD-IDF is that it will tend to give recommendations that are just that - the mean of preferred items. This can easily be a problem when our user's interest sits on opposite sides of the spectrum.

KNN: It follows instance based learning, so it has no training period. New data can be added seamlessly without losing accuracy. It is easy to implement as only two parameters are required for its implementation i.e K and distance. Even with all these advantages, it has certain disadvantages. It does not work well with large datasets and higher dimensions. At points, it needs dimension scaling. It is also sensitive to noisy data and outliers.

Neural Networks: To solve all these problems, neural networks come to the rescue. They work well with large sets of data with higher dimensions. They do not need dimension scaling. Noisy data, missing values and outliers do not affect the results. Neural networks also figure out the hidden trends in the data resulting in better results. The only drawbacks include that it takes a lot of computational power which is something we are willing to trade off.

VII. Future Work

In the future we can incorporate the movie posters into our dataset. We can implement more algorithms for both content based and collaborative filtering, for example SVD, CNN. In future we can incorporate the movie posters into our dataset. We can use convolutional neural networks (CNNs) to compare all the movie posters and find similarities between them. Then we can use some ensemble method to incorporate these results into our main results. This would result in a much more personalized experience for the user. We also intend to use other python libraries as well as see how the ALS Spark improves recommendation.

VIII. Conclusions

We have implemented both content based and collaborative filtering. We make recommendations to new users using Neural Networks and N-nearest neighbors algorithms. Also, for existing users using Natural Language Processing and Term Frequency (TF) and Inverse Document Frequency (IDF). We worked on many different strategies, few of them had errors best for us were the one explained above.It was a learning curve and a great experience. We learnt a lot with this project and hope to use what we learned to solve problems in future as well.

[9]. [4]. [8]. [7]. [10]. [2]. [11]. [5]. [1]. [6]. [3].

References

[1] Ebru Arisoy, Tara N Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 20–28. Association for Computational Linguistics, 2012.

[2] Chumki Basu, Haym Hirsh, William Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *Aaai/iaai*, pages 714–720, 1998.

[3] Sonia Bergamaschi and Laura Po. Comparing lda and lsa topic models for content-based movie recommendation systems. In *International conference on web information systems and technologies*, pages 247–263. Springer, 2014.

[4] Md Tayeb Himel, Mohammed Nazim Uddin, Mohammad Arif Hossain, and Yeong Min Jang. Weight based movie recommendation system using k-means algorithm. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1302–1306. IEEE, 2017.

[5] Youchun Ji, Wenxing Hong, Yali Shangguan, Huan Wang, and Jing Ma. Regularized singular value decomposition in news recommendation system. In *2016 11th International Conference on Computer Science & Education (ICCSE)*, pages 621–626. IEEE, 2016.

[6] Abhishek Kumbhar, Mayuresh Savargaonkar, Aayush Nalwaya, Chengqi Bian, and Mohamed Abouelenien. Keyword extraction performance analysis. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 550–553. IEEE, 2019.

[7] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[8] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[9] Dewi Soyusiawaty and Yahya Zakaria. Book data content similarity detector with cosine similarity (case study on digilib. uad. ac. id). In *2018 12th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pages 1–6. IEEE, 2018.

[10] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[11] Anand Shanker Tewari, Naina Yadav, and Asim Gopal Barman. Efficient tag based personalised collaborative movie reccommendation system. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 95–98. IEEE, 2016.