Movies and Video Series recommendation using emotional information.

ISEP supervisor

Professor: Raja CHIKY

By: Rakesh NUVVULA

61088

Table of Contents

1. Objective:

Creating dataset using given Emotional dataset and IMDB dataset to building recommended system.

2. About datasets

2.1 Emotional Dataset:

Emotional dataset contains 3659 records of data. It contains emotional scores for the movies and series episodes. It contains scores for 20 different emotions (Aggressiveness, Anger, Anticipation, Awe, Contempt, Disgust, Disapproval, Fear, Joy, Love, Negative, Optimism, Positive, Remorse, Sadness, Submission, Surprise, Trust, AFINN (-4&-5), Ero).

2.2 TheMovie Dataset:

TheMovie dataset have files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

belongs_to	budget	genres	homepage	id	imdb_id	original_la	original_tit	overview	popularity	poster_pa	production	production	release_da	revenue	runtime	spoken_la	status	tagline	title	video	vote_aver	vote_count
{'id': 1019	30000000	[{'id': 16, '	http://toys	862	tt0114709	en	Toy Story	Led by Wo	21.94694	/rhIRbceoI	[{'name': '	[{'iso_316	########	3.74E+08	81	[{'iso_639_	Released		Toy Story	FALSE	7.7	5415
	65000000	[{'id': 12, 'name': 'Adv		8844	tt0113497	en	Jumanji	When sibli	17.01554	/vzmL6fP7	[{'name': '	[{'iso_316	########	2.63E+08	104	[{'iso_639_	Released	Roll the di	Jumanji	FALSE	6.9	2413
{'id': 1190	0	[{'id': 10749, 'name': '		15602	tt0113228	en	Grumpier (	A family w	11.7129	/6ksm1sjK	[{'name': '	[{'iso_316	########	0	101	[{'iso_639_	Released	Still Yelling	Grumpier (	FALSE	6.5	92
	16000000	[{'id': 35, 'name': 'Cor		31357	tt0114885	en	Waiting to	Cheated o	3.859495	/16XOMpE	[{'name': '	[{'iso_316	########	81452156	127	[{'iso_639_	Released	Friends are	Waiting to	FALSE	6.1	34
{'id': 9687	0	[{'id': 35, 'name': 'Cor		11862	tt0113041	en	Father of t	Just when	8.387519	/e64sOI48	[{'name': '	[{'iso_316	########	76578911	106	[{'iso_639_	Released	Just When	Father of t	FALSE	5.7	173
	60000000	[{'id': 28, 'name': 'Acti		949	tt0113277	en	Heat	Obsessive	17.92493	/zMyfPUeI	[{'name': '	[{'iso_316	########	1.87E+08	170	[{'iso_639_	Released	A Los Ange	Heat	FALSE	7.7	1886
	58000000	[{'id': 35, 'name': 'Cor		11860	tt0114319	en	Sabrina	An ugly du	6.677277	/jQh15y5Y	[{'name': '	[{'iso_316	########	0	127	[{'iso_639_	Released	You are cc	Sabrina	FALSE	6.2	141
	0	[{'id': 28, 'name': 'Acti		45325	tt0112302	en	Tom and H	A mischiev	2.561161	/sGO5Qa5	[{'name': '	[{'iso_316	########	0	97	[{'iso_639_	Released	The Origin	Tom and H	FALSE	5.4	45
	35000000	[{'id': 28, 'name': 'Acti		9091	tt0114576	en	Sudden De	Internatio	5.23158	/eoWvKD6	[{'name': '	[{'iso_316	########	64350171	106	[{'iso_639_	Released	Terror goe	Sudden De	FALSE	5.5	174
{'id': 645, '	58000000	[{'id': 12, '	http://ww	710	tt0113189	en	GoldenEye	James Bon	14.68604	/5c0ovjT4	[{'name': '	[{'iso_316	########	3.52E+08	130	[{'iso_639_	Released	No limits.	GoldenEye	FALSE	6.6	1194
	62000000	[{'id': 35, 'name': 'Cor		9087	tt0112346	en	The Ameri	Widowed	6.318445	/lymPNGL	[{'name': '	[{'iso_316	########	1.08E+08	106	[{'iso_639_	Released	Why can't	The Ameri	FALSE	6.5	199

Figure 1: Sample TheMovie Data

For our project we were using following files from TheMovie dataset

1. movies_metadata.csv for IMDB movie data
2. links_small.csv to merge emotional dataset and TheMovie dataset.

3. credits.csv for movies cast and crew data.
4. keywords.csv for the keywords used in movies.
5. ratings_small.csv for movielens ratings.

3. Cleaning Dataset

3.1 Removing unwanted data.

The emotional dataset contains many unwanted columns which were not required for the project, so they were removed manually. The removed data columns were as below LoveA, OptimismA, AgressivenessA, ContemptA, SubmissionA, AweA, DissaprovalA, RemorseA, erowords and nrc words.

3.2 Adding IMDB ids to Emotional data.

Titles of the emotional data movie and tv-series are not same as titles in IMBD. Emotional data titles contain -, .srt for every movie and some titles are separated by ".", "white space" as shown in the below image.

Figure 2 About Emotional data titles

So, titles were not used to merge two datasets and there is no other way to merge. So IMDB ID's (tconst) are assigned to emotional data titles manually.

	IMBD-tconst	0	Score	Normalise	Year
0	tt12389600	12 Angry Men 1957 en	8.9	87.23404	1957
1	tt0047478	1954 Seven Samurai	8.6	80.85106	1954
2	tt12409982	1984 Once Upon a Time in America Il etait une fois en Ameriq	8.3	74.46809	1984
3	tt0062622	2001 A Space Odyssey 1968 REMASTERED 720p BluRay X264 /	8.3	74.46809	1968
4	tt1187043	3 Idiots 2009 Hindi 1080p Blu Ray x264 DD 5 1 MSubs HDSect(	8.3	74.46809	2009
5	tt0066921	A Clockwork Orange (1971) 1080p H 264 Multi (moviesbyrizzo	8.3	74.46809	1971

Figure 3 Emotional dataset with IMDB id's

3.3 Cleaning TV-series data of Emotional dataset

Emotional data contains data of Tv-series episodes but there is only one IMDB id for complete series and for recommended system we cannot give recommendations by episode. To overcome this problem, the emotional scores of each Tv-series episode were calculated manually.

Boardwalk Em	8.6	80.8511	2010	2148.241	840.017241	50.8930586	63.306953	13.21022296	22.9929	34.914888	13.589	22.92337
Boston Med S(	8.6	80.8511	2010	3164	1083	37.6941607	60.903251	4.059095106	15.8834	52.965668	23.818	7.9841509
Boston Med S(	8.6	80.8511	2010	2980	1000	35.8562612	71.5881	3.297	15.5658	49.44	8.5985	13.363291
Boston Med S(	8.6	80.8511	2010	3170	1110	39.8471547	62.561953	3.96036036	17.7613	44.438984	10.328	25.494355
Boston Med S(	8.6	80.8511	2010	3114	1108	41.3942191	68.149225	4.959386282	12.8067	52.924188	12.934	16.317461
Boston Med S(	8.6	80.8511	2010	3162	1131	41.9065255	64.043421	3.886825818	18.037	46.666667	12.671	19.460766
Boston Med S(	8.6	80.8511	2010	2689	971	42.8410383	64.078342	9.054582904	15.7257	55.315045	13.775	20.238825
Boston Med S(	8.6	80.8511	2010	2673	990	45.3770831	60.704819	2.22020202	16.1207	44.565657	5.7902	19.850403
Boston Med S(	8.6	80.8511	2010	2603	1033	52.6215241	68.11875	6.383349468	8.92136	51.42128	7.3989	8.3706055
Boston Med S(	8.6	80.8511	2010	2944.375	1053.25	42.1922458	65.018483	4.727600245	15.1028	49.717186	11.914	16.384982

Figure 4 Average of emotional scores for Tv-series

The average scores of tv-series, movies data were taken into another file to build a clean and reduced dataset.

4. Merging Emotional dataset and TheMovie dataset

4.1 Merge with *movies_metadata.csv*

The movies_metadata.csv is the key files in the TheMovie dataset. It cantains *genres, id, imdb_id, original_language, original_title, overview, popularity, poster_path, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, title, video, vote_average, vote_count.*

There are IMDB id as common column in both datasets. So, the merge operation in performed on both datasets using the following code.

```
data = pd.read_csv('emotionalData.csv')
data = data.rename(columns={'tconst': 'imdb_id'})

theMovie = pd.read_csv('movies_metadata.csv')

 merge = pd.merge(data,theMovie, on ='imdb_id')

merge.to_csv('mergedData.csv')
```

Figure 5. Data Merge.

In the emotional dataset, imdb ids are given in the tconst column. so, the column name is renamed as imdb_id. Then, the merge operation is performed on emotional dataset and movies_matadata dataset. result of the merge is stored into a new csv file called "*mergeData.csv*"

5. Recommended System:

Considering the dataset, 4 different types of recommended systems were built as below.

1. Simple Recommender based on Emotional score.

2. Content Based Recommender.

    - Movie Overviews and Taglines
    - Movie Cast, Crew, Keywords and Genre

3. Collaborative Filtering.

    - using K-nearest neighbours
    - using singular-Value Decomposition

4. Hybrid Recommender with emotional scores.

To find the highest emotion score in a movie *get_emotionalScore* function was developed.

```
def get_EmotionScore(df):
    df1 = df[["Agressiveness", "Anger","Anticipation", "Awe", "Contempt", "Disgust", "Dissaproval", "Fear", "Joy",
    df1 = df1.astype(int)
    df3 = df1.idxmax(axis = 1)
    df2 = list(df3)
    return df2[0]
```

Figure 6: Get Emotional score

This function takes all emotion score columns and finds the highest score row using *idxmax*.

and returns that column index name.

Weighted rating: TMDB Ratings were used to come up with our Top Movies Chart. IMDB's weighted rating formula was used to construct chart. Mathematically, it is represented as follows:

$$\text{Weighted Rating (WR)} = (v.v+m.R)+(m.v+m.C)$$

where,

v - number of votes for the movie.

m - minimum votes required to be listed in the chart.

R - average rating of the movie.

C - mean vote across the whole report.

```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

Figure 7: weighted rating

5.1 Simple Recommender based on Emotional score.

The Simple Recommender offers generalized recommendations to every user based on movie popularity and emotional score. The basic idea behind this recommender is that movies that are more popular and more critically acclaimed will have a higher probability of being liked by the average audience. This model does not give personalized recommendations based on the user.

The implementation of this model is extremely trivial. The movies ware sorted based on emotional score, ratings and popularity and display the top movies in list. As an added step, emotion name was passed as an argument to get the top movies of a particular emotion.

```
emotionalData_build('Agressiveness').head(5)
```

	title	Year	IMDB-rating	weighted_Rating	vote_count	vote_average	popularity	Agressiveness
17	City Lights	1931	8.5	7.962115	444	8.2	10.891524	47.370690
7	Amadeus	1984	8.3	7.923136	1107	7.8	12.677592	40.008172
56	Scarface	1983	8.2	7.963056	3017	8.0	11.299673	26.732432
89	The Usual Suspects	1995	8.5	7.999228	3334	8.1	16.302466	26.341695
88	The Sting	1973	8.2	7.940888	639	7.9	12.016821	26.294434

Figure 8. Simple Recommender based on Emotional score.

5.2 Content Based Recommender.

A content-based recommender works with data that the user provides, either explicitly (rating) or implicitly. Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

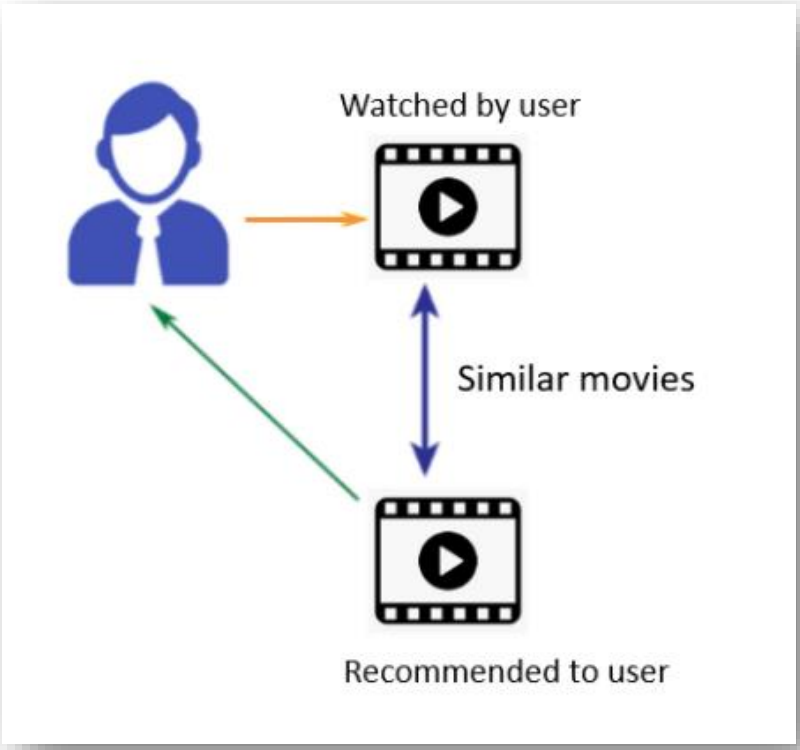

Figure 9: content-based system.

This recommended system engine that computes similarity between movies based on certain metrics and suggests movies that are most similar to a particular movie that a user liked. Since we will be using movie metadata (or content) to build this engine, this also known as Content Based Filtering.

The recommender we built above suffers some severe limitations. It gives the same recommendation to everyone, regardless of the user's personal taste. If a person who loves

aggressiveness movies (and hates *love*) were to look at our Top 15 Chart, s/he wouldn't probably like most of the movies.

Two Content Based Recommenders were build based on:

- Movie Overviews and Taglines
- Movie Cast, Crew, Keywords and Genre

content based recommender system using movie Overviews and Taglines:

Here taglines, overviews were considered. using **TfidfVectorizer**, the total numbers of words in taglines and overviews were counted and tokenized into a vector. The implementations is as mentioned in below figure.

```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(smd['description'])
```

Figure 10: TF – IDF vectorizer

[***Tf-idf Vectorizer***: The Tf-idf Vectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. ... A vocabulary of 8 words is learned from the documents and each word is assigned a unique integer index in the output vector.]

Cosine Similarity

Cosine Similarity is used to calculate a numeric quantity that denotes the similarity between two movies. Mathematically, it is defined as follows:

$$cosine(x,y)=x.y^T/||x||.||y||$$

Since TF-IDF Vectorizer was calculated, calculating the Dot Product will directly give us the Cosine Similarity Score. Therefore, we will use sklearn's linear_kernel instead of cosine_similarities since it is much faster.

```
cosine_sim[0]
```

```
array([1.        , 0.00805253, 0.        , 0.        , 0.        ,
       0.        , 0.01238417, 0.        , 0.        , 0.        ,
       0.        , 0.00609925, 0.        , 0.03148928, 0.        ,
       0.02409024, 0.        , 0.00885412, 0.        , 0.        ,
       0.00903198, 0.        , 0.        , 0.0176804 , 0.        ,
       0.        , 0.        , 0.01465997, 0.01249017, 0.        ,
       0.00848464, 0.        , 0.        , 0.        , 0.02448305,
       0.        , 0.        , 0.        , 0.00673802, 0.00828134,
       0.02040472, 0.        , 0.        , 0.        , 0.        ,
       0.02240047, 0.        , 0.        , 0.        , 0.00525777,
```

Figure 11: Cosine similarity

Pairwise cosine similarity matrix was created for all the movies in our dataset. The next step is to write a function that returns the similar movies based on the cosine similarity score.

```
get_recommendations('The Dark Knight').head(10)
```

```
66                     The Dark Knight Rises
78                              The Departed
20                          Django Unchained
1                      2001: A Space Odyssey
19                                  Das Boot
92             Witness for the Prosecution
37                       Life Is Beautiful
2                                  3 Idiots
77                 The Silence of the Lambs
72    The Lord of the Rings: The Two Towers
Name: title, dtype: object
```

Figure 12: recommended system based on overview and tags.

For "*The Dark Knight*", our system is able to identify it as a Batman film and subsequently recommend other Batman film as its top recommendations. But unfortunately, that is all this system can do at the moment. This is not of much use to most people as it doesn't take into

considerations very important features such as cast, crew, director and genre, which determine the rating and the popularity of a movie. Someone who liked "*The Dark Knight"* probably likes it more because of Nolan and would hate Batman movies.

Therefore, much more suggestive metadata was considered below than Overview and Tagline. Will build a more sophisticated recommender that takes genre, keywords, cast and crew into consideration.

Content based: Metadata Based Recommender:

We now have cast, crew, genres and credits, all in one data-frame. The data is wrangle as shown below:

Crew: From the crew, only picked the director as feature since the others don't contribute that much to the feel of the movie.

Cast: Lesser-known actors and minor roles do not really affect people's opinion of a movie. Therefore, only selected the major characters and their respective actors. so, choose the top 3 actors that appear in the credits list.

The plan is creating a metadata dump for every movie which consists of genres, director, main actors and keywords. Then use a *Count Vectorizer* to create count matrix as did in the above Recommender. The remaining steps were similar to earlier: Calculate the cosine similarities and return movies that are most similar.

The following steps were followed in the preparation of genres and credits data:

- Strip Spaces and Convert to Lowercase from all features. This way, recommended engine will not confuse between Johnny Depp and Johnny Galecki.
- Mention Director 3 times to give it more weight relative to the entire cast.

The following code performs the steps mentioned above.

```
smd['cast'] = smd['cast'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
smd['cast'] = smd['cast'].apply(lambda x: x[:3] if len(x) >=3 else x)
```

```
smd['keywords'] = smd['keywords'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
```

Figure 13: cast and keywords conversion.

The results for the above code is as follows

```
smd['cast']
```

```
0           [toshirōmifune, takashishimura, yoshioinaba]
1           [keirdullea, garylockwood, williamsylvester]
2                    [aamirkhan, kareenakapoor, madhavan]
3         [malcolmmcdowell, patrickmagee, adriennecorri]
4      [golshiftehfarahani, shahabhosseini, taranehal...
                             ...
94            [milesteller, j.k.simmons, melissabenoist]
95       [tyronepower, marlenedietrich, charleslaughton]
97         [samworthington, zoesaldana, sigourneyweaver]
100              [hughgrant, andiemacdowell, jamesfleet]
101                                    [davidattenborough]
```

Figure 14: cast output

The top three cast names are selected into an array.

Keywords

A small amount of pre-processing of keywords before putting them to any use. As a first step, calculate the frequent counts of every keyword that appears in the dataset.

```
s = smd.apply(lambda x: pd.Series(x['keywords']),axis=1).stack().reset_index(level=1, drop=True)
s.name = 'keyword'
```

```
s = s.value_counts()
s[:10]
```

```
based on novel    11
violence           8
world war ii       7
dystopia           7
nazis              5
friends            5
revenge            5
corruption         5
murder             5
love               5
Name: keyword, dtype: int64
```

Figure 15:most relevant Keywords

The outputs shown in above figure are the most repeated keywords in the dataset. Keywords occur in frequencies ranging from 1 to 11. there is no use of any keywords that occur only once. Therefore, these can be safely removed. Finally, convert every word to its stem so that words such as *Humans* and *Human* are considered the same.

Snowball Stemmer: It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the Porter Stemmer since some issues of it were fixed in this stemmer.

input------> output
cared ----> care
fairly ----> fair
singing ----> sing
sings ----> sing
sung ----> sung
singer ----> singer
sportingly ----> sport

With the help of snowball stemmer, CountVectorizer, cosine_similarity the keywords are processed and stored into a matrix. The data is all set for the recommended system.

```
get_recommendations('The Dark Knight').head(10)
```

```
66       The Dark Knight Rises
74                 The Prestige
38                      Memento
32                 Interstellar
86           The Usual Suspects
53                      Scarface
51                Reservoir Dogs
77    The Silence of the Lambs
63                  Taxi Driver
78                 The Departed
Name: title, dtype: object
```

Figure 16: content-based recommender using metadata

now, the results are much better than the previous recommended system. The movies *"The dark knight", "The drak knight raises", "The prestige", "Memento", "intersteller"* has same director and some matches crew members.

5.3 Collaborative Filtering

The content based engine within built previous suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

Also, the engine is not personal in that it doesn't capture the personal tastes and biases of a user. Anyone query engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who s/he is.

Therefore, the technique called Collaborative Filtering to make recommendations to Movie Watchers. Collaborative Filtering is based on the idea that users similar to a me can be used to predict how much I will like a particular product or service those users have used/experienced, but I have not.

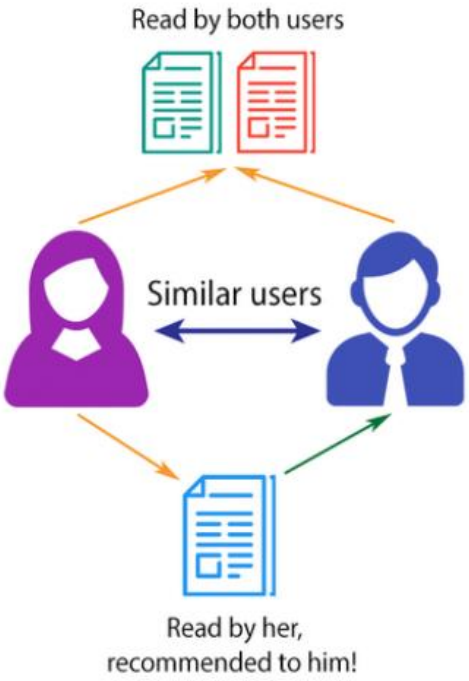

Figure 17: collaborative filter

The implementation of Collaborative Filtering is not from scratch. Instead, used Surprise library that used extremely powerful algorithms like *"Singular Value Decomposition"* (SVD), *"k-nearest neighbours" KNN-basics* to minimise "*RMSE (Root Mean Square Error)"* and give great recommendations.

Now, the rating from movielens come to play. using "Kfold" from surprise, the data is split into 5 number of folds where each fold is used as a testing set at some point.

Two different algorithms were used to train and predict the rating for the movie for user which the user haven't watched.

1. Using K-nearest neighbours:

Using Cross_validate from the surprise, the k-nearest neighbours' algorithm is performed for split data.

```
# We'll use the k-nearest neighbor algorithm.
knn = KNNBasic()
# Run 5-fold cross-validation and print results
cross_validate(knn, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Figure 18: K-nearest neighbours

The above code will calculate the *"Root Mean Square Error [RMSE]"*, and the "*Mean Square Error [MSE]".*

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9654	0.9741	0.9653	0.9626	0.9764	0.9688	0.0054
MAE (testset)	0.7423	0.7489	0.7441	0.7411	0.7477	0.7448	0.0030
Fit time	0.11	0.12	0.12	0.13	0.13	0.12	0.01
Test time	0.98	1.01	1.11	1.05	1.04	1.04	0.04

Figure 19: RMSE & MSE using KNN-Basics.

The RMSE using K-nearest neighbours is 0.9688. It is more than enough for prediction.

Now, we can train the data using KNNBasics and predict the results.

```
trainset = data.build_full_trainset()
knn.fit(trainset)
```

Figure 20: train data using KNN-basics.

The above command trains the dataset with KNNBasics.

```
knn.predict(1, 2, 3)

Prediction(uid=1, iid=2, r_ui=3, est=3.2584390074346286,
```

Figure 21: movie predict using KNN-Basics.

In the above figure, the prediction is performed using KNNBasics for the movie id: 2 by the user 1 by degree 3 and the estimated rating is 3.258.

```
knn.predict(2, 2, 3)

Prediction(uid=2, iid=2, r_ui=3, est=3.3101434365546045,
```

Figure 22: movie predict using KNN-Basics.

For the same movie prediction is performed by the used 2 and the estimated rating is 3.310

2. Using* singular-Value Decomposition (SVD)*:

procedure is same as the K-nearest neighbours for the singular value decomposition. The Root Mean Square Error, Mean Square Error of the data using the SVD algorithm is:

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8854  0.8910  0.8934  0.9117  0.9009  0.8965  0.0091
MAE (testset)     0.6856  0.6888  0.6885  0.6995  0.6904  0.6906  0.0047
Fit time          15.42   16.08   14.98   15.11   15.66   15.45   0.39
Test time         0.40    0.50    0.38    0.51    0.35    0.43    0.06
```

Figure 23: RMSE, MSE using SVD

The RMSE using SVD is: 0.8965

```
svd.predict(1, 2, 3)

Prediction(uid=1, iid=2, r_ui=3, est=2.3965355676800644,
```

Figure 24: movie predict using SVD

In the above figure, the prediction is performed using SVD for the movie id: 2 by the user 1 by degree 3 and the estimated rating is 2.3965

```
svd.predict(2, 2, 3)

Prediction(uid=2, iid=2, r_ui=3, est=3.2856199737395446,
```

Figure 25: movie predict using SVD

For the same movie prediction is performed by the used 2 and the estimated rating is 3.2856.

5.4 Hybrid Recommender with emotional scores

The Hybrid recommender system is the combination of content based and collaborative recommended systems, which are built above.

- *Input*: User ID and the Title of a Movie
- *Output*: Similar movies sorted on the basis of expected ratings by that particular user and highest emotional score for the input movie

```
hybrid(1, 'The Dark Knight')
```

	title	vote_count	vote_average	Year	id	Trust	est
76	The Shawshank Redemption	8358.0	8.5	1994	278	41.551177	3.672766
67	The Godfather	6024.0	8.5	1972	238	60.690334	3.518884
86	The Usual Suspects	3334.0	8.1	1995	629	38.776227	3.510090
78	The Departed	4455.0	7.9	2006	1422	63.377656	3.473601
88	To Kill a Mockingbird	676.0	7.9	1962	595	54.418776	3.434179

Figure 26: Hybrid recommender example 1

As we seen in the above figure, the engine captured the movie name which are based on both collaborative filter and content based recommended system and the highest emotional scores in the "The dark knight", the movies recommended all have good *"trust"* emotional score.

```
hybrid(2, 'The Dark Knight')
```

	title	vote_count	vote_average	Year	id	Trust	est
67	The Godfather	6024.0	8.5	1972	238	60.690334	4.427138
47	Psycho	2405.0	8.3	1960	539	44.815462	4.404246
49	Rear Window	1531.0	8.2	1954	567	27.561833	4.334965
74	The Prestige	4510.0	8.0	2006	1124	53.969619	4.309208
51	Reservoir Dogs	3821.0	8.1	1992	500	33.492096	4.234994

Figure 27: Hybrid recommender example 2

For same movie as input, we can see a lot of different movies recommended for the user 2

6. learnings:

- Although I have knowledge on python, I have never worked on any real-time project like this. This project helped me to gain more knowledge on python.
- I learned how Netflix, amazon prime work to recommend movies.
- I got really good knowledge on movie recommended system and how to implement.
- I got knowledge on datasets like IMDB, movielens, TMDB, TheMovie.

7. References:

https://surprise.readthedocs.io/en/stable/getting_started.html

https://www.imdb.com/interfaces/

https://grouplens.org/datasets/movielens/

https://www.kaggle.com/rounakbanik/the-movies-dataset

https://blog.codecentric.de/en/2019/07/recommender-system-movie-lens-dataset/

https://nbviewer.jupyter.org/github/oekosheri/Recommender-movie/blob/master/Recom_walk_through.ipynb

https://medium.com/@jdwittenauer/deep-learning-with-keras-recommender-systems-e7b99cb29929