CS550: Movie Recommendation system

Prakruti Joshi, Twisha Naik, Hari Priya Ponnakanti, Dhruv Mundhra
Rutgers University
Email: phj15, tn268, hp467, dm1396@scarletmail.rutgers.edu

***Abstract*—**Recommendation systems are ubiquitous in versatile online platforms these days. Our aim is to build a movie recommendation system based on 'MovieLens' dataset. We wish to integrate the aspects of personalization of user with the overall features of movie such as genre, popularity etc. We have implemented popularity model, content based model, collaborative filtering model and latent factor based model. The hyperparameter tuning, testing accuracy and evaluation of recommendations of each model are thoroughly performed. We combine the predictions of latent factor based and collaborative filtering method to form a combined linear model which shows an improvement in the accuracy of the predicted ratings. To generate the final recommendation list for each user, we combine all the methods to form a hybrid model to overcome the limitations of individual models. The hybrid model performs better than the individual models in terms of quality and diversity of recommendations. Detailed analysis of each model is given in the report.

Key Words - movie recommendation, content-base, collaborative filtering, latent factor, SVD, hybrid recommendation

I. Introduction and Related Work

With the large amount of growth in data, and capturing the consumer behaviour, one of the most important aspects of today's technologies is providing personalized services.

The two basic approaches of recommendation systems are Collaborative Filtering and Content Based Filtering [FermanFerman2002]. In year 2005, Adomavicius [Adomavicius G.Adomavicius G.[n.d.]] came up with a hybrid approach, merging both methods therefore reducing the limitations of them individually. Today, hybrid approach is the most commonly used method for recommendation systems.

SVD or Singular Value Decomposition is one such commonly used algorithm that is used to perform Collaborative Filtering. SVD performs Matrix Factorization on the user-movie matrix to generate the corresponding decomposition vectors showing similarity between movies.

II. Data

a) About Data:
The primary dataset used for this project is the *grouplens movie review dataset*. The smaller dataset *(ml-latest-small)* describes 5-star rating and free-text tagging activity from *MovieLens*, a movie recommendation service. The dataset has a collection of 27,753,444 reviews over 58,098 different movies by 283,228 users. Each rating being a value between 0-5. Some features of the dataset include the timestamp of the review, genre of the movie, the keywords of comments by the users, and the IMDB and TMDB id for the corresponding movie.

b) Data preprocessing:

- We split the dataset into test and train on the basis of UserID, such that 80% of the reviews by every single user is used for training the model and the remaining 20% of the reviews to validate the correctness of our model. We achieve the above using the train test split method in scikit-learn library. The stratify attribute of the function specifies the feature based on which the dataset is divided into testing and training.

- To gain a better intuition about the movie for content based recommendation and importance of features of movies, we integrated the data set with the publicly available IMDB and TMDB dataset, leading to features such as release date, writer, director, and cast information, tag line, overview, popularity of movie, the rating and vote counts on their respective websites.

- Cleaning of data: Some cleaning process included removing redundant and non-contributing features from the dataset. Converting genres and other categorical features to one-hot vector. We also excluded any users having less than 5 data points, since we need at least 1 data point for testing.

c) Data Analysis

a) Genre Distribution of Movies

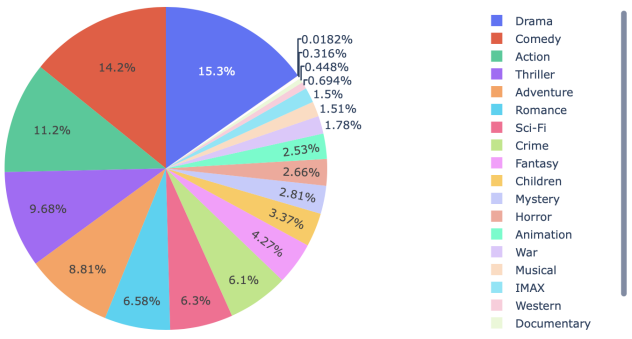

Fig. 1: Genre distribution in training data

b) Average number of user ratings per movie = 477
c) Average number of movies rated by user = 98
d) User Rating Distribution

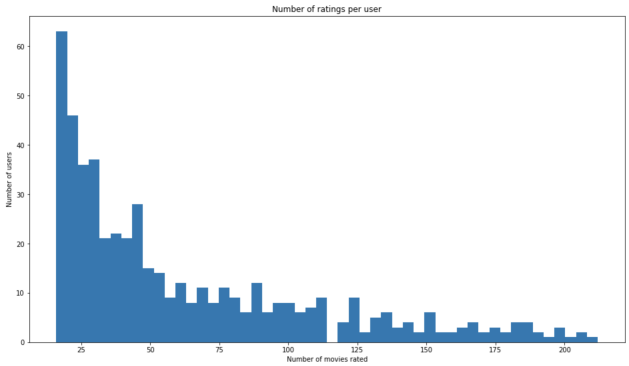

Fig. 2: Histogram for number of ratings per user

e) Rating Distribution:

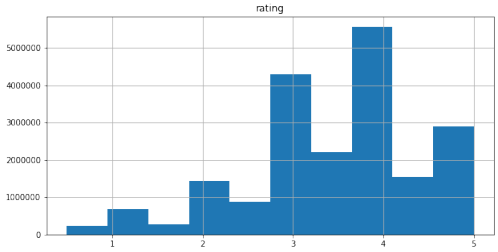

Fig. 3: Histogram for ratings

III. Model

1) **Content based Recommendation**

Content based recommendation system works on the idea of finding similar items based on the content of items. In case of movies, we tried to include the main features of movies like genres, release year, overview and tagline. The following two approaches were tried:

- **Generating user vector and item vector**

  Movies can be well described by the genre it falls under. And this property is again used to prepare a user vector.

  Movie vector = Vector of zeros and ones with a one for a relevant genre

  User vector = Average rating of the user for that particular genre

Item Profile			
Movie ID	**Action**	**Romance**	**Comedy**
1	1	0	0
2	0	1	1
3	0	1	0
Ratings			
User ID	**Movie ID**	**Rating**	
1	1	2	
1	2	5	
1	3	4	
2	1	4	
2	3	2	
User Profile			
User ID	**Action**	**Romance**	**Comedy**
1	2	4.5	5
2	4	2	0

Fig. 4: Sample example for user and movie profile generation

Metric	**Content-based (Genre)**
RMSE	0.9185
MAE	0.7095

TABLE I: RMSE vs MAE Evaluation

Metric	**Content-based (Genre)**
Precision	0.80093
Recall	0.49516
F-Measure	0.61198
NDCG	0.94558

TABLE II: Other Evaluation Metrics

- **Movie-Movie Similarity**

  On combining the MovieLens data with TMDB data, we extracted the information of movie overview and taglines. Once we have this, TF-IDF vectorizer was applied and a TF-IDF vector was generated for each movie.

  After the generation of TF-IDF vector, cosine similarity was calculated for each movie-movie movie pair.

  On analysing the results of using just the content in movie description, we found it was completely ignoring the genre and was just focusing on the number of common words in the description. As it is clear from the example below, the movies having Christmas and Santa appear in the similar list.

```
get_recommendations('Doctor Who: Last Christmas')
```

```
314                          The Santa Clause
16254                How I Ended This Summer
42025                             Santa Claus
46467                The Spirit of Christmas
22527    The Life & Adventures of Santa Claus
2316                  Miracle on 34th Street
37762       The Christmas That Almost Wasn't
25725                               Santa Who?
40212    The Life & Adventures of Santa Claus
```

Fig. 5: Movie Similarity based on overview and tagline

Thus we also added the genre information while constructing the movie description. Again, adding genre just once did not change the results. Hence, we added this information twice.

Now, by the inherent property of TF-IDF vectorizer, it reduces the weight of terms frequently occurring over the entire document. Thus, as the names of genre will appear in all the movies, it will be given lesser importance. Thus, we changed the TF-IDF vectorizer to count vectorizer and repeated the experiments.

```
get_recommendations_new('Doctor Who: Last Christmas')
```

```
30647                          20 Years After
19000                  4:44 Last Day on Earth
38193    Wizards of Waverly Place: The Movie
36355                    Under the Mountain
10418                        Born in Flames
45510                             Team Thor
28                 The City of Lost Children
20474               It's Such a Beautiful Day
27268                    Master of the World
40678                             On the Comet
```

Fig. 6: Movie Similarity after adding genre information

Movie	Genre	Overview
'Doctor Who: Last Christmas'	['Adventure', 'Drama', 'Fantasy', 'Sci-Fi']	'The Doctor and Clara face their Last Christmas. Trapped on an Arctic base, under attack from terrifying creatures, who are you going to call? Santa Claus!'
'20 Years After'	['Drama', 'Fantasy', 'Sci-Fi']	'In the middle of nowhere, 20 years after an apocalyptic terrorist event that obliterated the face of the world!'
'4:44 Last Day on Earth'	['Drama', 'Fantasy', 'Sci-Fi']	'A look at how a painter and a successful actor spend their last day together before the world comes to an end.'

TABLE III: Analysis of top two similar movies

As it can be seen from the results in Fig. 6, the similar movies are no longer based on the description but also take into consideration the genre.

As it can be seen from the table III the overview of the movies are similar depicting something negative like attack and apocalypse. On the other hand, it also takes into consideration the similarity of genre.

2) **Collaborative Filtering**

Collaborative filtering uses the ratings to form a neighbourhood N of a user, say X, whose ratings (likes and dislikes) are similar to X's ratings. The neighborhood of X is defined using nearest neighbor approach with a notion of similarity defined. There can be two types of collaborative filtering user-user and item-item. This method is different from content based as the neighborhood or similarity is defined only on the basis of rating behaviour. We have used the Surprise library prediction algorithms to analyse the performance of KNN (K-nearest neighbor) algorithm for collaborative filtering. The following are the experiments done for collaborative filtering algorithm.

Algorithm	RMSE	MAE
Baseline Only	1.06123	0.82837
KNNBasic	1.19982	0.93565
KNNBaseline (Item-Item)	**0.98966**	**0.75491**
KNNBaseline (User-User)	1.10828	0.85270
SVD	1.05494	0.81750
SVDPP	1.00836	0.78535
Content based	1.0584	0.794279

TABLE IV: Performance (RMSE and MAE) of algorithms when user has rated less than 18 movies. (Cold start problem)

a) **Variations of KNN based models**:

- *KNNBasic*: A basic collaborative filtering algorithm which takes the maximum number of neighbors *k* to take into account and the similarity metric as parameters.
- *KNNwithMeans*: A basic collaborative filtering algorithm similar to *KNNBasic* which takes into account the mean ratings of each user.
- *KNNWithZScore*: A basic collaborative filtering algorithm similar to *KNNBasic* which takes into account the z-score normalization of each user.
- *KNNBaseline*: A collaborative filtering algorithm which takes into account a baseline rating.

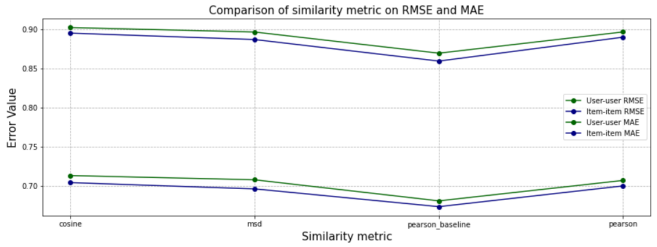

Fig. 7: Comparison of user-user and item-item

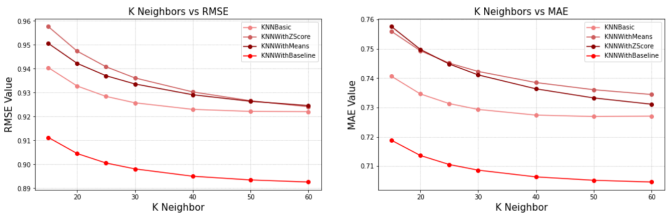

Fig. 8: Effect of number of neighbors on RMSE and MAE for methods involving KNN

The performance of the variations of the model can be observed in Fig. 8

b) **Similarity metrics**:

- Cosine similarity: Cosine similarity is a measure of similarity between two non-zero vectors.
- Mean squared difference: The mean squared similarity is defined as inversely proportional to the difference of mean squared difference.
- Pearson correlation: Pearson correlation coefficient can be interpreted as a mean-centered cosine similarity.
- Pearson Baseline: This method computes the Pearson correlation coefficient between all pairs of users (or items) using baselines for centering instead of means.

c) **User-user vs Item-item similarity**: The collaborative filtering method can be implemented for user-user similarity where the neighborhood of user X is defined by users who have given similar ratings to the items rated by user X. The dual of this is item-item similarity where the neighborhood of item I is items who have ratings similar to item I which are rated by common users.

We analysed the performance of KNN based algorithms for item-item and user-user similarity.

- The item-item based collaborative filtering sur-

passes user-user based collaborative filtering in all the variations of KNN algorithm. This can be clearly observed in **Fig. 7**. The results are so since the notion of similarity is more stronger in items than in users. For example, items can be categorized into genres and few items have overlapping genres. On the other hand, user can have varied and unique tastes which can be quantified easily.

- The item-item based collaborative filtering also has an advantage of performing better in cases where a new user is introduced to the system (cold start problem). This can be seen in **Table IV**.

d) Number of Neighbors: We experimented with the maximum number of neighbors to take into account for aggregation. As the number of neighbors increases, the accuracy of the methods also increases. This can be observed in **Fig.8**

After all these experiments we find that the ***KNNBaseline*** algorithm for item-item similarity and the similarity metric as ***Pearson baseline*** gives the best results for our movie dataset5)

3) **Matrix Factorizatin methods**

We want our recommendation model to predict the unknown ratings. For this we use the latent factor method, *SVD*. SVD takes the lower dimensional representation of movies such that people who like similar movies together are mapped together. It discovers the features (hidden latent factors) such that movies can be mapped into the same space as user. We can use the SVD model to form an optimization problem to minimize the RMSE for unseen test data. Thus, using gradient descent algorithm and regularization allows rich model structure.

To model biases and interactions, we can use a baseline predictor which takes into account the user bias b_x, the item or movie bias b_i and the overall mean rating μ into its rating estimation. Adding these biases improves the results.

We have used the Surprise library to analyse the matrix factorization models. We have used ***GridCVSearch*** from sklearn to find the hyperparameters which give the lowest RMSE. The best hyperparameters are:

- Number of epochs (n_epochs): 10
- Number of latent factors $(n\_factors)$: 50
- Regularization parameter (λ) : 0.02
- Learning rate for all parameters (lr_all): 0.01

SVDpp: This model is an extension of SVD which takes into account implicit ratings. An implicit rating describes the fact that a user u rated an item j, regardless of the rating value. This addition of implicit ratings along with explicit ratings gives the best results. However, SVDpp takes the longest time to train and fit and is much slower than other simpler models.

The testing accuracy and the recommendation evaluation results are displayed in **Table VI**.

4) **Combined model (SVD + CF)**

From the **Table VI**, we can observe that the following methods give the best results in terms of testing accuracy and recommendation evaluation:

- KNNBaseline (with pearson baseline as similarity)
- SVDpp
- SVD
- BaselineOnly (takes the baseline predictor means of global ratings, user and the item/movie)

To improve the test accuracy results and balance the limitations of each method, we applied a linear combination of the ratings from the above methods. KNNBaseline helps to introduce the item similarity notion for a user, SVDpp introduces the implicit ratings along with the latent factor model. SVD balances the over-estimations of SVDpp. BaselineOnly helps to introduce the global biases of user and the items.

Combining these models improves the RMSE, MAE and the recommendation evaluations. This can be clearly observed in **Table V**.

5) **Popularity model**

On merging the given MovieLens data with TMDB data, we get a new feature of popularity defined by the collective activity of the users such as average rating, number of views, likes, favourites, watchlist additions and release date. Using this information, we generated a genre wise popular movie list which can be used while recommendation for a user with a preferred preference.

Apart from popularity, we have also considered the weighted ratings. The weighted ratings can be computed using the following equation:

$$W = R\frac{v}{v+m} + C\frac{m}{v+m}$$

where,

W = Weighted Rating

R = Average rating of a movie (scale: 1-10)

v = number of votes for the movie

m = minimum votes required to be listed in top

C = Mean vote average

SVD	KNNBaseline (PearsonBaseline)	SVDPP	SlopeOne	Baseline	RMSE	MAE	Precision	Recall	F-Measure	NDCG
0	0	1	0	0	0.8626	0.6735	0.79921	0.33642	0.47352	0.95872
0	1	0	0	0	0.8527	0.6482	0.81984	0.36242	0.50264	0.96310
0	0.6	0.4	0	0	0.844	0.6427	0.82732	0.35428	0.49611	0.96637
0.05	0.6	0.35	0	0	0.84405	0.6427	0.82836	0.35424	0.49626	0.96731
0.1	0.5	0.3	0	0.1	0.84405	0.6433	0.83139	0.35040	0.49301	0.96614

TABLE V: Performance of the model for weighted combination of the best algorithms

```
genre_based_popularity('Animation')[['title', 'popularity']]
```

	title	popularity
32753	Minions	547.488298
25016	Big Hero 6	213.849907
47929	Captain Underpants: The First Epic Movie	88.561239
5522	Spirited Away	41.048867
48707	Despicable Me 3	36.631519
45789	A Silent Voice	34.852055
44363	Your Name.	34.461252
50057	The Emoji Movie	33.694599
4793	Monsters, Inc.	26.419962
39518	Zootopia	26.024868
6275	Finding Nemo	25.497794
21452	Despicable Me 2	24.823550

Fig. 9: Popular movies for the genre "Animation"

6) **Hybrid model**

Hybrid model predicts ratings based on the combination model which worked best so far along with the user's preference for a genre and popularity model. Till now we were only looking at the movies in the test data for recommending it to the user. The issue with this approach was suppose a user just has 4 movies in the test data and we want 5 recommendations for that user. In this case, regardless of the rating, the movie was getting recommended. We wanted to avoid recommending low rated movies and instead look for movies which neither the user has seen nor rated.

The proposed flow of the hybrid model is as shown in the Figure: 10.

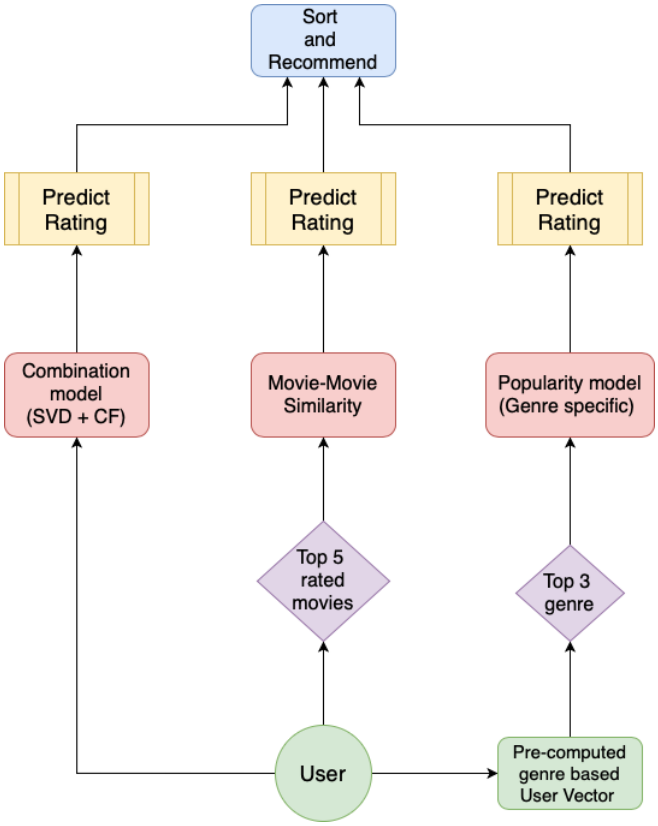

Fig. 10: Flow of the Hybrid recommendation model

As shown in the figure, given a user we perform the following steps:

a) Combination model:
Initially predict the ratings for all the movies in the test set.

b) Movie - Movie Similarity model:
We find the top 5 highly rated movies by the user from the training data and extract similar movies to that.

c) Genre Popularity model:
From the user vector created using the genre, we find the top 3 genres the user likes and get the 5 most popular movies from each genre.

Once we have the ratings of all these movies, we sort it decreasing order of estimated ratings and recommend top 10 movies to the user.

We analyse this model for two types of users. Ones with high number of ratings and other with low ratings. The results of the hybrid model can be seen in the images below:

Case-1: The user has enough training and testing data

In this case, the value of estimated ratings for each movie is very high. And the top movies were already included in the test set.

Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller
4.39	4.65	4.48	4.27	4.24	4.33	4.53	4.33	4.39	4.05
Horror	Mystery	Sci-Fi	War	Musical	Documentary	IMAX	Western	Film-Noir	Other
3.58	4.28	4.19	4.61	4.63	0	0	4.4	5	0

Fig. 11: Genre based user vector

movieId	est	Model	title	genre
2571	5.000000	SVD + CF	[The Matrix]	[['Action', 'Sci-Fi', 'Thriller']]
1208	5.000000	SVD + CF	[Apocalypse Now]	[['Action', 'Drama', 'War']]
608	4.998963	SVD + CF	[Fargo]	[['Comedy', 'Crime', 'Drama', 'Thriller']]
2542	4.989595	SVD + CF	[Lock, Stock and Two Smoking Barrels]	[['Comedy', 'Crime', 'Thriller']]
5618	4.925268	Popularity	[Spirited Away]	[['Adventure', 'Animation', 'Fantasy']]
2078	4.875510	Popularity	[The Jungle Book]	[['Animation', 'Children', 'Comedy', 'Musical']]
2186	4.846480	Popularity	[Strangers on a Train]	[['Crime', 'Drama', 'Film-Noir', 'Thriller']]
364	4.835412	Popularity	[The Lion King]	[['Adventure', 'Animation', 'Children', 'Drama...
1748	4.832581	Popularity	[Dark City]	[['Adventure', 'Film-Noir', 'Sci-Fi', 'Thrille...
1252	4.820373	Popularity	[Chinatown]	[['Crime', 'Film-Noir', 'Mystery', 'Thriller']]

Fig. 12: Hybrid model recommendations

Case-2: The user does not have enough ratings in the training and testing data

In this case, the popularity model comes into picture. Based on the genre preference of the user, list of most popular movies are extracted and then ratings are predicted for those movies. If only the movies in test set were recommended, then we miss out on few of the movies which the user would have really liked.

Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller
3.875	0	0	3.67	5	3.17	3.58	3.2	3.2	2.78
Horror	Mystery	Sci-Fi	War	Musical	Documentary	IMAX	Western	Film-Noir	Other
1.75	4	3.17	3	3	0	0	4	3.9	0

Fig. 13: Genre based user vector

movieId	est	Model	title	genre
174053	4.103651	Popularity	[Black Mirror: White Christmas]	[['Drama', 'Horror', 'Mystery', 'Sci-Fi', 'Thr...
5965	3.981930	SVD + CF	[The Duellists]	[['Action', 'War']]
1201	3.945507	Popularity	[The Good, the Bad and the Ugly]	[['Action', 'Adventure', 'Western']]
1283	3.936192	Popularity	[High Noon]	[['Drama', 'Western']]
168366	3.925894	Popularity	[Beauty and the Beast]	[['Fantasy', 'Romance']]
54997	3.925103	Popularity	[3:10 to Yuma]	[['Action', 'Crime', 'Drama', 'Western']]
5618	3.921863	Popularity	[Spirited Away]	[['Adventure', 'Animation', 'Fantasy']]
187	3.907950	SVD + CF	[Party Girl]	[['Comedy']]
175197	3.903956	Popularity	[The Dark Tower]	[['Fantasy', 'Horror', 'Sci-Fi', 'Western']]
171759	3.902344	Popularity	[The Beguiled]	[['Drama', 'Thriller', 'Western']]

Fig. 14: Hybrid model recommendations

IV. Experiments and Results

1) **Framework:**
We have coded our project in Python and storing and processing data using Pandas dataframe. For latent factor methods and KNN based algorithms, we used Surprise library [HugHug2017] for experimenting with different prediction algorithms and evaluating our models. Surprise is a Python scikit library used for building and analyzing recommender systems that deal with explicit rating data.

2) **Evaluation parameters:**
We split the data into two by considering 80% of the data for each user as training data while 20% of the data to be testing data. We evaluated the ratings predicted by our model by using the RMSE and MAE evaluation metrics. We also used other metrics like Precision, Recall, F-Measure and NDCG to evaluate the recommendations returned by the model. We set the rating threshold t to be 3.75 to identify the recommended and relevant items used in the calculation of Precision and Recall metrics. Recommended items are those items with predicted rating greater than or equal to t while relevant items are those with actual rating greater than or equal to t. We used the model to recommend the top 10 and top 5 movies to each user and observed that the model performed better in the case of recommending top 5 movies.

3) **Popularity model:**
Model returns the most popular movies for a given genre based on two metrics:
1. Popularity feature in TMDb
2. Weighted rating as defined by IMDb.

4) **Model experiments:**

- **Content based model:**
  - Derived user profile based on the item profile considering genre and year of the release of movie.
  - RMSE: 0.9185 and MAE: 0.7095
    The results can be seen in Table I.
  - Developed a movie-similarity model considering the following features of the movies:
    - Genre
    - Tag line
    - Overview
- **Collaborative filtering model:**
  - Variations of KNN based algorithms:
    KNNBaseline gives the best results while generating prediction for testing data in terms of RMSE and MAE. It also gives the best recall value while generating recommendations and evaluating using precision and recall @ top-5. The results can be seen in **Table VI**.
  - Similarity metric analysis:
    Pearson Baseline gives the best results when combined with KNNBaseline algorithm. The result can be seen from **Fig.7**
  - User-user vs Item-Item Analysis: Item-item based collaborative filtering has better accuracy than user-user based collaborative filtering as shown in **Fig. 7**. It even performs better when the user has less number of ratings in the training data. This can be observed from **Table IV**.
  - The results improve as we increase maximum neighbors in the KNN algorithms. This can be seen from **Fig. 8**.
- **Matrix Factorization model:**

Algorithm	RMSE	MAE	fit_time	test_time	Precision	Recall	F-measure	NDCG
KNNBaseline(pearson_baseline)	**0.852705**	**0.64818**	9.18326	6.47820	**0.83117**	0.41316	**0.55196**	**0.96310**
KNNWithZScore	0.89918	0.67862	0.14657	1.79509	0.79480	0.39152	0.52462	0.95420
KNNWithMeans	0.90003	0.68369	0.10967	1.54060	0.80096	0.38714	0.52198	0.95271
KNNBasic	0.95077	0.72665	**0.09878**	1.38924	0.78388	**0.42153**	0.54824	0.95868
KNNBaseline	0.87629	0.66599	0.21010	1.93655	0.79642	0.41588	0.54642	0.95622
BaselineOnly	0.87346	0.67176	0.13966	**0.09251**	0.80743	0.40035	0.53528	0.95908
SlopeOne	0.90564	0.68769	4.75922	5.5923	0.80754	0.39652	0.5318	0.9555
SVDpp	**0.86912**	**0.66405**	480.69708	7.99730	**0.81784**	0.39788	0.53532	**0.96032**
SVD	0.87944	0.67394	4.60998	0.12506	0.80330	0.38554	0.52102	0.95660
NMF	0.92914	0.70946	5.15004	0.20406	0.77928	0.38074	0.51155	0.95488
CoClustering	0.95221	0.73326	1.82520	0.17341	0.78262	0.38098	0.51248	0.95574

TABLE VI: Analysis for Latent Factor methods and CF methods implemented using surprise library

  - SVDpp is one of the best method which gives th lowest predicting error. However, it is much slowe method in terms of training.
  - Hyperparameter tuning of SVD parameters usin GridSearchCV improves the testing prediction re sults.
  - The testing accuracy and the evaluation parameter can be seen in **Table VI**.
- **Combined model:**
  - The comparison of various KNN based and matri factor based methods can be seen in Fig. 15, Fig.16 and Fig.17. A weighted linear combination of the top methods gives better testing accuracy of ratings and even outperforms the best individual models (KNNBaseline and SVDpp). The NDCG values improve incase of the combined model. We experimented with the different weights for the linear combination of the models - KNNBaseline, SVDpp, SVD, BaselineOnly. The weights assigned to each model and the respective evaluation results are shown in Table **Table V**.
- **Hybrid model:**
  - We integrated the above models into a single model to generate recommendation lists. Currently, the combined model (SVD + CF) recommended movies from the testing set. We have merged the top movies from popularity model, combined model(SVD + CF), content based (movie similarity) and chosen the top movies based on estimated ratings as the final recommendation list. Though the model cannot be evaluated using standard parameters since we do not have the original ratings, the quality and diversity of recommendations has improved. The results improve for users which have rated less movies validating that the hybrid model is an efficient solution to the cold start problem often faced in recommendation systems.

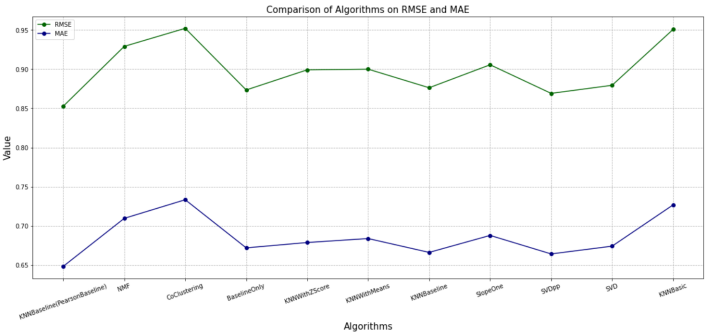

Fig. 15: Comparison of algorithms based on MAE RMSE

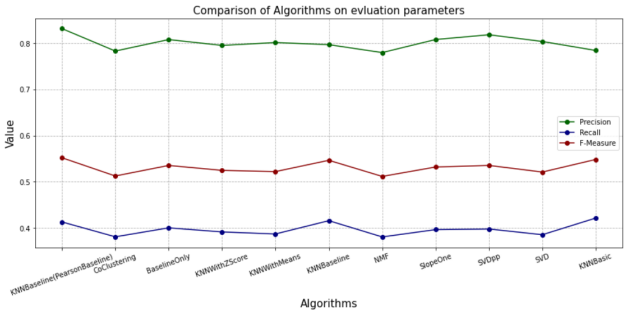

Fig. 16: Precision and Recall values for different methods

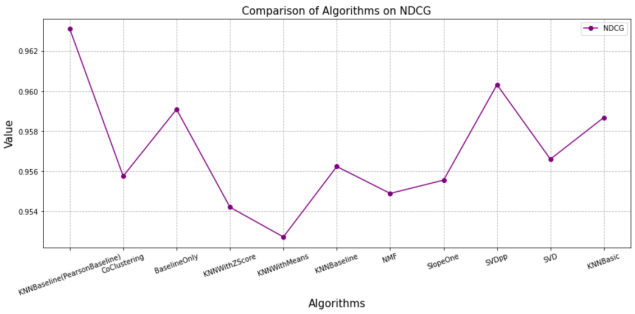

Fig. 17: NDCG values for different methods

V. Conclusion

The content based and popularity based model are decent models for generating recommendations and understanding the notion of similarity. They are useful to solve the cold start problem for a new user. However, they lack personalization

for a given user. Collaborative filtering and latent factor methods capture the user related features well and are more powerful for large datasets. Combining the models of CF and SVD improves the accuracy of the predicted ratings as seen in the combined model. The hybrid model extracts the features from each model and helps to create a balanced recommendation list for a user.

VI. Future Work

A possible extension of the hybrid model is creating a feature vector for model and training a neural network to learn the importance or the weights of each model. Such a linear combination model can highly improve the recommendations. Learning to rank using deep learning methods is another possible extension. Currently, we have implemented the models using the smaller dataset. Extending this to the larger dataset and observing the results is yet another future task. Another possible extension in case of content based recommendation is adding the director and cast information to generate the item and user profiles.

Acknowledgement

We would like to thank the Professor Mr. Yongfeng Zhang and the teaching assistant Mr. Shuyuan Xu for their guidance as well as for providing necessary information regarding the project and also for their support in completing the project.

References

[Adomavicius G.Adomavicius G.[n.d.]] A. Adomavicius G., Tuzhilin. [n.d.]. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.. In *Regional Conference Series in Mathematics*.

[FermanFerman2002] Errico J. H. Beek P. V. Sezan M. I. Ferman, A. M. 2002. Content-based filtering and personalization using structured metadata.. In *Regional Conference Series in Mathematics*.

[HugHug2017] Nicolas Hug. 2017. Surprise, a Python library for recommender systems. http://surpriselib.com.

VII. APPENDIX

(A) Rating Evaluation parameters

- **Root Mean Square Error (RMSE):**
  Root Mean Square Error is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data, that is, how close the observed data points are to the model's predicted values. It is a frequently used measure of the differences between values predicted by a model or an estimator and the values observed. The smaller an RMSE value, the closer the predicted and observed values are or the closer you are to finding the line of best fit. RMSE is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE.
  Formula to calculate RMSE:

  $$\text{RMSE} = \sqrt{(\tfrac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$$

  where,
  n is the number of data points,
  y_i is the predicted value,
  x_i is the actual/observed value

- **Mean Absolute Error:**
  The Mean Absolute Error measures the average magnitude of the errors in a set of predictions, without considering their direction. The MAE is a linear score which means that all the individual differences are weighted equally in the average. It tells us how big of an error we can expect from the prediction model on average. The MAE will always be lesser than or equal to MAE. If all the errors have equal magnitude, then MAE = RMSE. The lower the value of MAE, the higher is the accuracy of the prediction model. Formula to calculate MAE:

  $$\text{MAE} = (\tfrac{1}{n})\sum_{i=1}^{n}|y_i - x_i|$$

  where,
  n is the number of data points,
  y_i is the predicted value,
  x_i is the actual/observed value

(B) Recommendation Evaluation parameters

- **Precision:**
  The set of k items which are recommended to each user is known as top-k set. Precision at k is the proportion of recommended items in the top-k set that are relevant, that is, it is the fraction of relevant items in the top-k set. It measures the ability of the prediction model to make relevant predictions. For example, if the Precision at 5 for a top-5 recommendation model is calculated to be 80%, it means that 80% of the recommendations made by this model are relevant to the user. Therefore, we always try to attain a higher value of precision for our model. Formula to calculate Precision:

  $$\text{Precision} = \frac{|\text{Recommended Items that are relevant—}|}{|\text{Recommended Items}|}$$

  where (with a threshold rating value of t),
  Recommended Items that are relevant are the items with both the original and predicted ratings greater than or equal to t,
  Recommended items are those items which have predicted rating greater than or equal to t.

- **Recall**
  Recall at k is the proportion of relevant items found in the set of top-k recommendations. For example, if the recall at 10 is calculated to be 35% for our prediction model, then it means that 35% of the total number of the relevant items appear in the top-10 recommendations. So, we always try to maximize the

value of recall for the model. Formula to calculate Recall:

$$\text{Recall} = \frac{|\text{Recommended Items that are relevant—}|}{|\text{Relevant Items}|}$$

where (with a threshold rating value of t),
Recommended Items that are relevant are those items with both the original and predicted ratings greater than or equal to t,
Relevant items are those items which have original rating greater than or equal to t.

- **F-measure**
The F-measure (also called F-score) is a measure of the prediction model's accuracy. It is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall). Formula to calculate F-measure:

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- **Normalized Discounted Cumulative Gain (NDCG):**
NDCG is a ranking metric which allows relevance scores in form of real numbers. It is used to evaluate the goodness of the recommendation list returned by the model. It is based on the concept that more relevant items should be placed at the beginning of the recommended list as items placed at later positions in the list tend to be overlooked by the users. Each item in the recommendation list has a relevance score which is the original rating in our case. This is called as gain G_i. For items which don't have a relevance score (or original rating), gain is usually set to zero. To see the most relevant items at the top of the list, we divide each by a growing number (usually a logarithm of the item position) which is called as discounting. We then add these discounted items to get a DCG (Discounted Cumulative Gain).

$$\text{DCG} = \Sigma_{i=1}^{n} \frac{rel_i}{log_2(i+1)}$$

To make DCGs directly comparable between users, we need to normalize them. We arrange all the items in the ideal order using the actual ratings and compute DCG for them which is called as Ideal DCG (IDCG). We then divide the raw DCG by this ideal DCG to get NDCG@K, a number between 0 and 1.

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}}$$

The worst possible DCG when using non-negative relevance scores is zero. Therefore, good recommendation lists have NDCG scores closer to 1.