Study of Recommender Systems

Arsh Irfan Modak, [1] Omkar Waghmare, [2] Manjit Ullal[3]

Northeastern University Khoury College of Computer Sciences [1,2,3]
modak.a@northeastern.edu, [1] waghmare.o@northeastern.edu, [2] ullal.m@northeastern.edu[3]

Abstract

Recommender systems are widely employed in the industry and are ubiquitous in our daily lives. The major aim of recommender systems is to help users discover new and relevant items such as movies to watch, text to read or products to buy, find compelling content and create a delightful user experience. There are three traditional methods: content, context and hybrid to implement recommender systems. Our goal is to present a comparative study between these traditional recommendation systems and their variants, along with neural networks; and hope to ameliorate some of the shortcomings of the traditional systems.

Introduction

There are three traditional methods (content-based, collaborative filtering and hybrid) to implement recommender systems. Our goal is to present a comparative study between the first two traditional methods and their variants.

Furthermore, we implement neural collaborative filtering in the hope to ameliorate some of the shortcomings of the traditional systems. Additionally, we will also experiment with Variational Autoencoders and Clustering to solve the problems.

Related Work

Traditional methods have been implemented on a smaller subset of the MovieLens 1 Million dataset as well as the MovieLens 100k dataset. Scikit-Surprise offers benchmark RMSE for both these datasets, however it does not offer a benchmark for the bigger 25 Million datasets.

Project Description

1. The Datasets:

The name of the dataset is "The Movies Dataset".[1] These 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, posters, revenue, 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 0.5-5. Additionally, we also use the MovieLens 1 Million and 25 Million datasets to implement the algorithms.[2]

2. Data Pre-Processing:

- To keep our data consistent with the Movie-Lens datasets we made sure all users rated at least 20 movies.
- The metadata file was really messy and needed a lot of cleaning to be used for our Context Based Recommender System.
- Columns such as Genres and Overview used to extract useful data from the movies metadata table
- Information such as Actors and Characters were extracted from the Cast Column from the Credits table.

3. Content-Based Recommender System

A Content-Based Recommender System recommends items based on similarity to other items a user has liked. If movie A and B are similar and a user has watched movie A, there is a high possibility movie B will be recommended to the user.

To Implement this, we extracted all the metadata available for the movies such as overview, actors, characters, genre etc., vectorized it using the TF-IDF Vectorizer and used the Cosine Similarity to generate a similarity matrix. This matrix was then used to recommend the top 10 movies to the user based on the item they liked.

4. Collaborative Filtering

Collaborative Filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users.[3] These reactions can be either explicit (ratings) or implicit (views, clicks, adding to wish list

etc.). For example, if two users U1 and U2 have a similar user profile and U1 likes a certain set of movies, it is highly possible that U2 will like some/all of those movies, hence, we can recommend U1 those movies.

To implement Collaborative Filtering, we used the Scikit-Surprise library in Python. For Movie-Lens 1M we implemented algorithms such as SVD, NMF, KNN based algorithms (Baseline, Basic, With Means, With Z-Score), Baseline Only, Normal Predictor, Slope One and Co-clustering.

For MovieLens 25M and "The Movies Dataset" we implemented SVD.

5. Limitations of Matrix Factorization

Today, neural networks are used for solving many business problems such as sales forecasting, customer research, data validation, and risk management. Our aim here is to see if we can utilize a neural network as the core of our collaborative model in order to work on implicit data.

Despite the popularity and effectiveness of matrix factorization for collaborative filtering, its performance was seriously affected by the "inner-product", which incurred in a large ranking loss.[4]

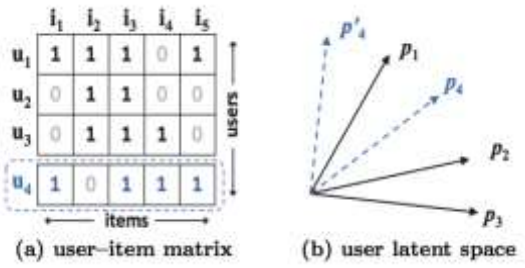


Figure 6.1[4]

The above example shows the limitation of the inner product to model the interactions between user and item latent space. Consider the first three rows of the matrix given in fig(a). On computing their similarities with each other we find out that $S_{23} > S_{12} > S_{13}$. This relationship can be easily demonstrated in the latent space. We can see that vector P_2 is closer to P_3, P_1 is closer to P_2 and P_1 is farthest from P_3. Their geometric interpretations have managed to maintain their similarity order. But, once we add another user U_4, this similarity relationship ($S_{41} > S_{43} > S_{42}$) cannot be maintained. If a model places P_4 closest to P_1 (the two options are shown in Figure 1b with dashed lines**),** it will result in P_4 closer to P_2 than P_3, which unfortunately will incur a large ranking loss.

6. Neural Collaborative Filtering

NCF[5] overcomes this drawback by utilizing the flexibility, non-linearity and complexity of neural nets to build a versatile recommender system[6]. This helps the model properly estimate the complex relations between the user and item latent vector space.

The architecture we have built for our neural collaborative filtering system is shown below.

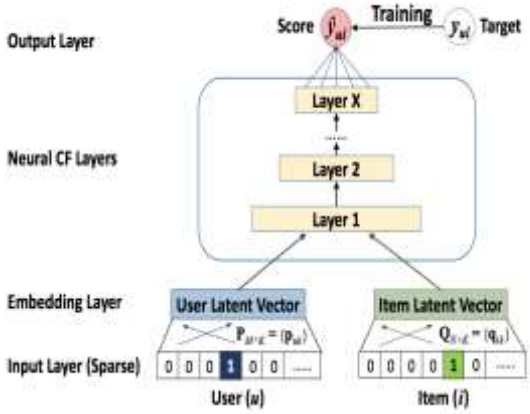


Figure 6.2[4]

Input Layer: Consist of sparse User and Item vectors.

Embedding Layer: Responsible for converting the user and item vectors in dense format. In our case, both user and item latent vectors are of dimension 8 x 1.

Neural CF Layers: Any configuration of deep neural networks works here. In our case we have 2 linear layers [(16,32), (32,64)] activated using the ReLU activation function.

Output Layer: The output of Neural CF layers goes as an input to the output layer. The output layer is responsible for predicting the score. In our case we have used a sigmoid activation function which outputs the probability of the item being interesting to the user or not.

Implementation

The code for neural collaborative filtering can be found at the GitHub repo under the NCF folder. "model_data" contains models and their required variables on 30,50,70 percent of unique users respectively. These have been included so that retraining of these models in not necessary and offline predictions can be quickly

accessed. The script for loading these checkpoints and running the models can also be found in the NCF folder.

Packages required: Torch, DataLoader, NumPy, Pytorch Lightning.

7. Variational Autoencoders

Majority of the collaborative-filtering models use latent factor models; however, these models are limited due to their use of linear nature. It has been shown that non-linear features improve a model's performance and predictive ability. Here, we implement the approach mentioned in the paper *Liang et al*, where the objective

is to minimize the top-N ranking loss which is approximated using multinomial likelihood.

Most recommender systems rely on explicit feedback. However explicit feedback is not always available. Therefore, recommender systems can learn from implicit feedback which are more abundant.

The latent representation Zu is transformed by a nonlinear function "f_θ" to produce a probability distribution over I items given the history of user interaction with X_u items. The output is normalized using SoftMax. The user rating is made as binary 0/1, which means either rater or not rated.

U = {1, 2, 3, … u} is the number of users
I = {1, 2, 3, … i} is the number of items/ movies

$\exp\{f_\theta(Z_u)\}$

Here X_u is assumed to be sampled from multinomial distribution with probability $\pi(Z_u)$.

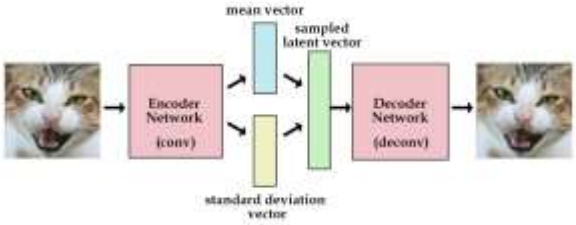
Log-Likelihood of the probability user rating given the data is:

$$\log p_\theta(x_u|Z_u) = \sum x_{ui} \log(\pi_i z_u)$$

Assuming Gaussian Log-Likelihood we get:

$$\log p_\theta(x_u|Z_u) = -\sum \frac{c_{ui}}{2}(x_{ui} - f_{ui})^2$$

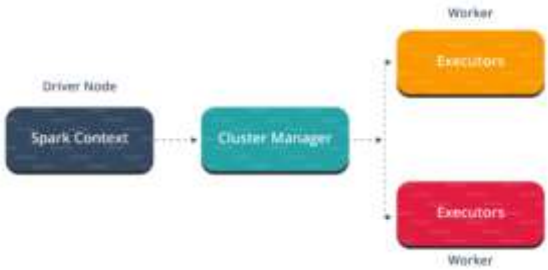
Here c is a weight introduced to balance the number of 0 for the user rating which is far more than the 1.



We learn the parameters of the generative model, using the variational distribution.

8. Recommendation using K-Means Clustering

Traditional recommendation suffers from the curse of dimensionality and requires high compute for big data. This problem can be mitigated by running the computation parallelly using a distributed framework like Spark.



The data is converted to Resilient Distributed Data (RDD) in Spark, which is distributed in memory and fault tolerant dataset using spark context And then the cluster manager will create more workers to execute the work and aggregate the results.
K-Means algorithm implemented in PySpark to perform clustering.

This technique can also be employed to perform matrix factorization like SVD.

Empirical Results

1. Content-Based Recommender Systems

We used various metadata to calculate similarity, namely, overview, overview + actors, overview + characters for recommending movies. The input was a movie name whose related movies would be the output. The output was sorted by similarity and the top k results are displayed. "k" can be set by the user.

We displayed the results for three movies: "Toy Story", "Star Wars" and "Jumanji" as shown in figures 3.

2. Collaborative Filtering using Surprise

Out of all the algorithms, SVD gave us the lowest RMSE of 0.8743 on the MovieLens 1M dataset as shown in figure 4.1. Hence, we stuck to SVD for our larger datasets. Since we had a maximum RAM of 25GB, we were not able to implement other algorithms on the larger datasets since the RAM was not enough for storing such large matrices. However, SVD seemed to work as it did not require more than 16.67GB for our largest dataset. We performed 5-fold cross validation for all algorithms.

The RMSE we got for the MovieLens 25 Million dataset and "The Movies Dataset" is 0.778 and 0.7815 respectively.

The comparison of the RMSE, Fitting and Testing times are shown in figures 4.2.

We also performed hyper-tuning of SVD on various parameters such as no. of factors, no. of epochs, learning rate for all attributes and regularization value for all the attributes. This process took 31 hours to complete. We noticed the resulting RMSE was higher than the RMSE we got from the default parameters even though they were included in the hyper-tuning module. We hope to

find out as to why this happened. Furthermore, hyper tuning SVD can sometimes go wrong and may give incorrect predictions, thus hits is a better measure of evaluation than RMSE for the algorithms. We will implement hits in the future for these algorithms as we have for Neural Collaborative Filtering.

3. Neural Collaborative Filtering

Pre-process the data: In the "Preprocess" class we read the data, subset it and split it into train and test sets.
Define model: In the "Net" class We define a neural net and its paraments. We run the model and train it on the train set.
Evaluation: In the "compute_hits" function we evaluate the model with hit ration at 10.
Recommendations: In the "recommend_user_movies" function we recommend movies to a certain user.

Model Output:

variable	30%	50%	70%
Our Performance (HIT ratio@10)	0.86	0.9	0.92
Time To Train (hours)	2	7	15
Number of Movies	43365	47634	52261
Number of Users	48762	81270	113778

Figure 6.3

Due to memory and compute power constraints, we were unable to run Neural Collaborative Filtering for the complete dataset.

4. Variational Autoencoders

The model will be evaluated on the top-N ranking loss, meaning for each user rank N movies using the multinomial probability these ranks will be compared to that of the actual rank.

All users are split into train/ validation/ test set and then the users are trained on their transaction history.

Two metrics are utilized to measure the loss. Recall and discounted cumulative gain.

Below NDCG is normalized discounted cumulative gain. Here the Rank_1 means the top 1 rank predicted by the model.

Model Output:

RECALL	NDCG
RECALL_1 : 0.4013	NDCG_1 : 0.4013
RECALL_5 : 0.3801	NDCG_5 : 0.3868
RECALL_10 : 0.3590	NDCG_10 : 0.3684
RECALL_20 : 0.3683	NDCG_20 : 0.3619
RECALL_50: 0.4567	NDCG_50 : 0.3898
RECALL_100: 0.5736	NDCG_100: 0.4347

5. Recommendation using K-Means Clustering

Optimal cluster that preserves maximum energy is found using elbow method. In our case it was k = 19, then the algorithm is run again to find the labels of each point. The intuition is that we can find similar movies for a movie by looking at the neighbors of the movie in the same cluster.

Conclusion and Future Work

- Hyper tuning for SVD can go wrong since the predictions are very sensitive to the hyper-parameters even if the model gives a lower RMSE
- HIT'S are a better measure for collaborative filtering evaluations than RMSE as we can know how relevant the predicted items are for a particular user than just predicting the ratings it would give to the item.
- Neural collaborative filtering utilizes the nonlinearity, complexity and flexibility of neural networks in order to better model user-item interactions.
- It works better than traditional models such as matrix factorization because it uses neural nets to learn user-item interactions and hence does not incur a ranking loss.
- Moving ahead, we aim to implement a GMF model that applies the linear kernel to model user-item interactions. We then combine these two models in order to combine the linearity of GMF and non-linearities of Deep Neural Nets for modelling user-item interactions.
- Implement SVD, Matrix Factorization using Spark.

- Majority of the recommendations systems are multi-modal and that they utilize multiple inputs and types of data to make inference. So we would like to experiment with a model that utilizes structured and unstructured information of a user for recommendation.

FIGURES AND OUTPUTS

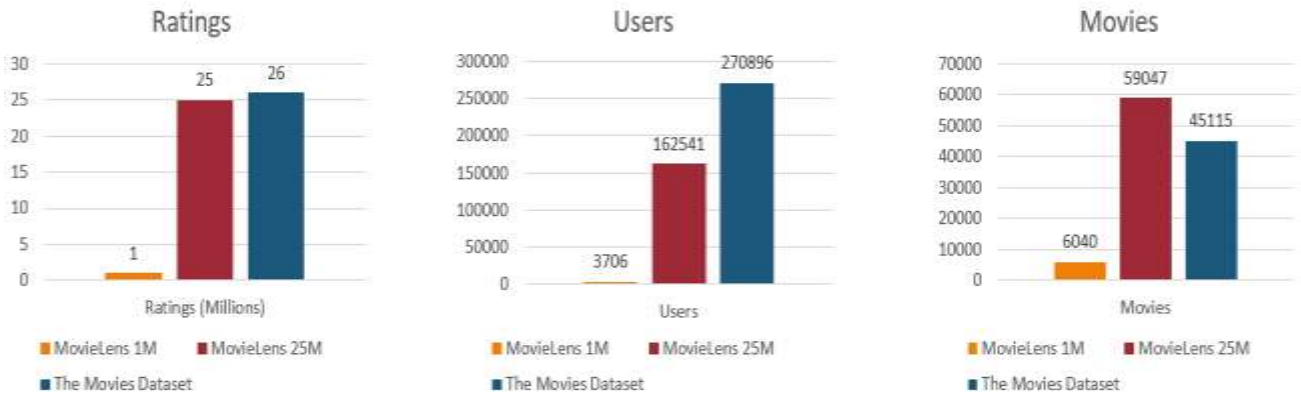


Figure 1: Distribution of Ratings, Users and Movies across all datasets.

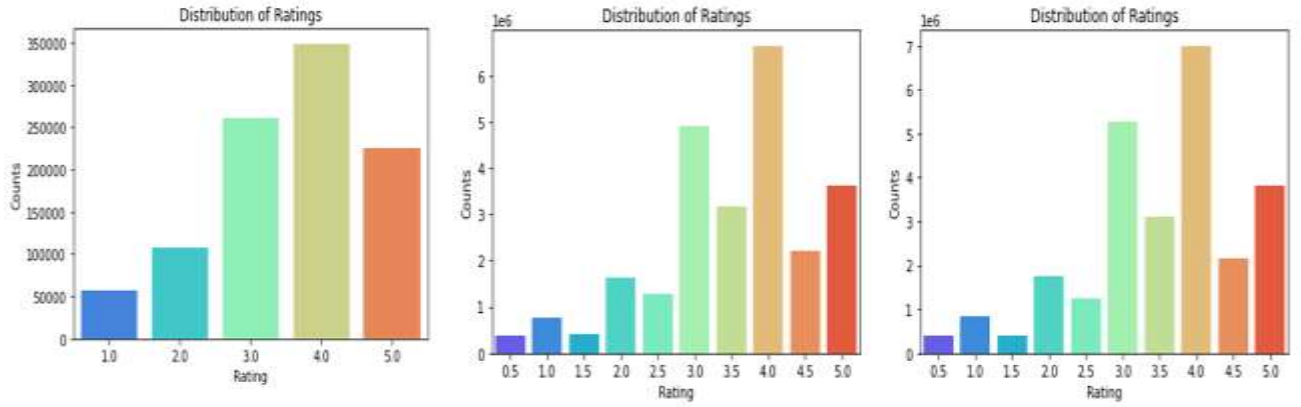


Figure 2: Distribution of Ratings over all datasets.

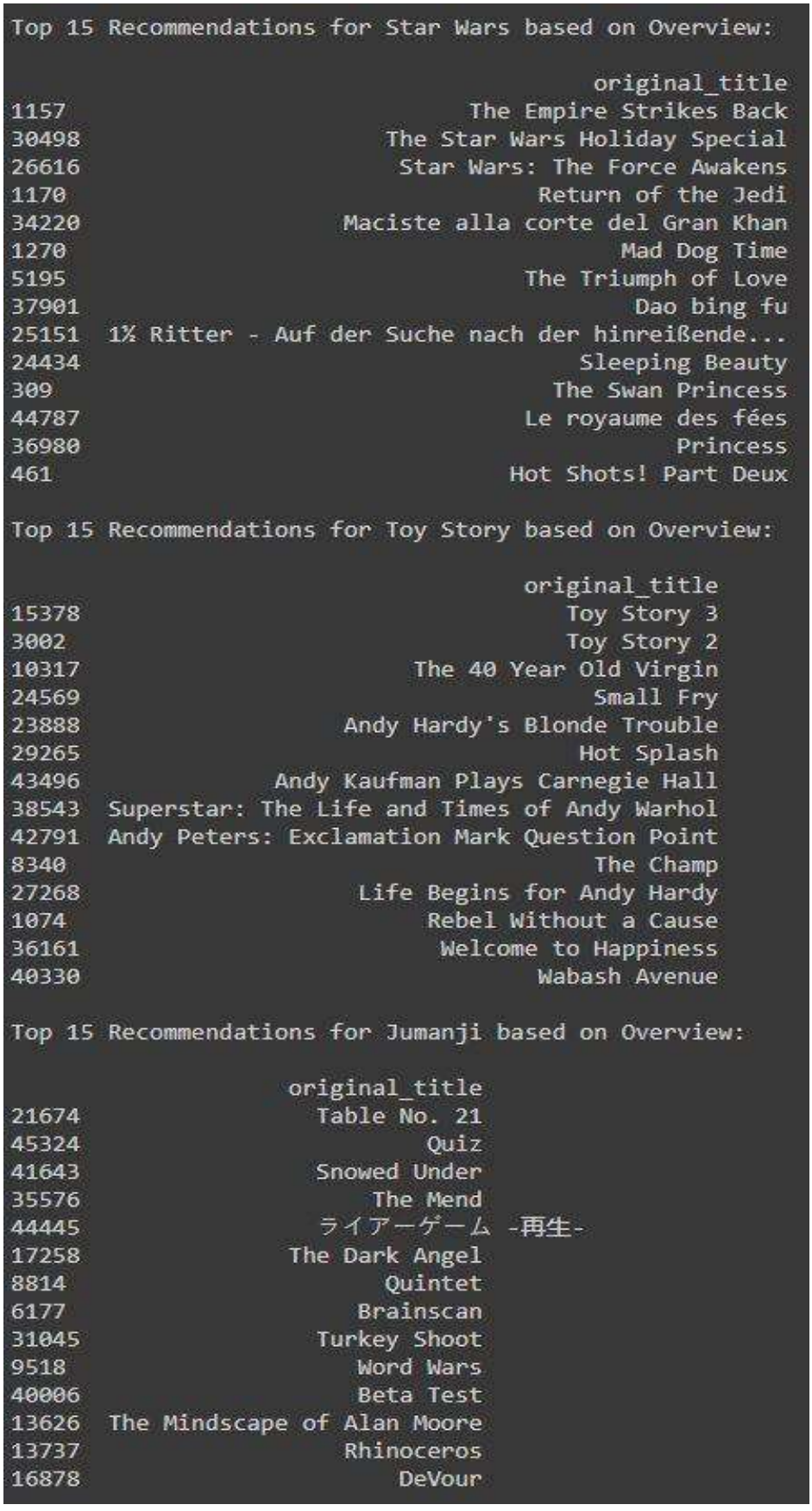


Figure 3: Top 15 Results of Content-Based Recommendation for Star Wars, Toy Story and Jumanji

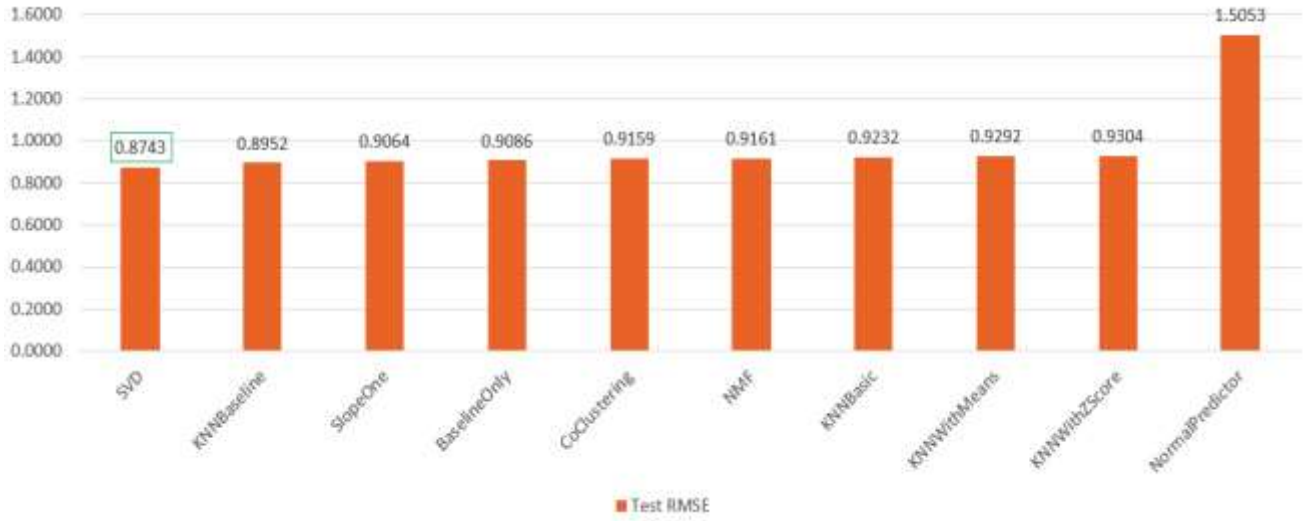


Figure 4.1: Comparison of RMSE of all algorithms on MovieLens 1 Million Data

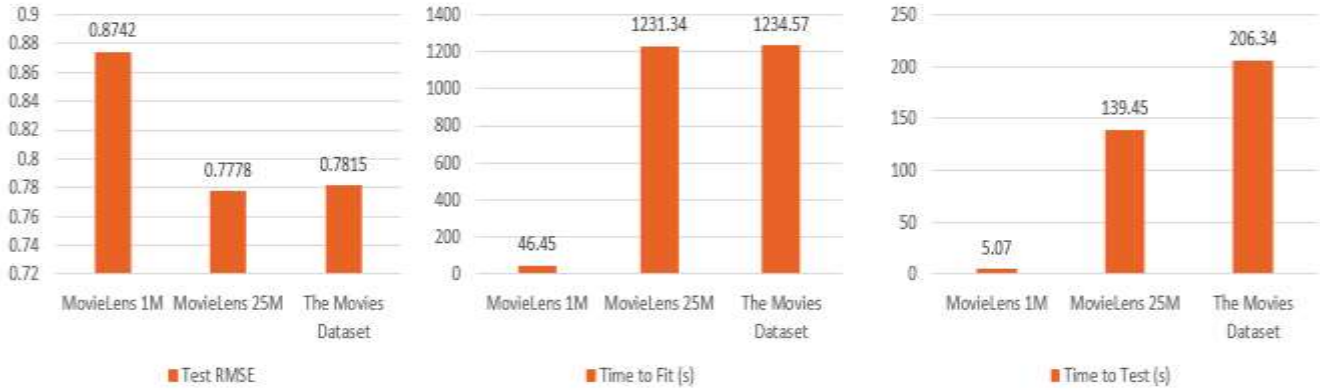


Figure 4.2: Comparison of Test RMSE, Fit Time and Train Time on all three datasets.

```
Predictions for UserId: 1
============================================================

                                              Movies   Ratings
2745                          American Beauty (1999)  4.658104
26660  Louis C.K.: Live at The Comedy Store (2015)  4.578219
36942        Requiem for the American Dream (2015)  4.567810
758                            Trainspotting (1996)  4.554061
4743                            Donnie Darko (2001)  4.538798
2287                                Rushmore (1998)  4.522521
40697                O.J.: Made in America (2016)  4.511944
42517                     Horace and Pete (2016)  4.506180
16241                Louis C.K.: Hilarious (2010)  4.502333
49                     Usual Suspects, The (1995)  4.496782
============================================================


Predictions for UserId: 54
============================================================

                                                    Movies   Ratings
45901                                               Cosmos  4.720490
45669                                Planet Earth II (2016)  4.718451
40537                                            Life (2009)  4.709211
2215          Life Is Beautiful (La Vita è bella) (1997)  4.688468
49480                                Blue Planet II (2017)  4.686354
5834        My Neighbor Totoro (Tonari no Totoro) (1988)  4.685242
12813                 Lonely Wife, The (Charulata) (1964)  4.680674
9565   Howl's Moving Castle (Hauru no ugoku shiro) (2...  4.648348
40536                                 Planet Earth (2006)  4.611287
5483   Spirited Away (Sen to Chihiro no kamikakushi) ...  4.606111
============================================================


Predictions for UserId: 777
============================================================

                                                    Movies  Ratings
1189      Seventh Seal, The (Sjunde inseglet, Det) (1957)      5.0
13758                               Thirst (Pyaasa) (1957)      5.0
31110                                  I, Claudius (1976)      5.0
28745              Sinatra: All or Nothing at All (2015)      5.0
20026      Great Beauty, The (Grande Bellezza, La) (2013)      5.0
9558   Howl's Moving Castle (Hauru no ugoku shiro) (2...      5.0
45174                                  Prohibition (2011)      5.0
8707                             Hearts and Minds (1974)      5.0
33537                                      Human (2015)      5.0
13696  Welcome Mr. Marshall (Bienvenido Mister Marsha...      5.0
============================================================
```

Figure 5: Results of Collaborative Filtering using SVD on MovieLens 25 Million Data

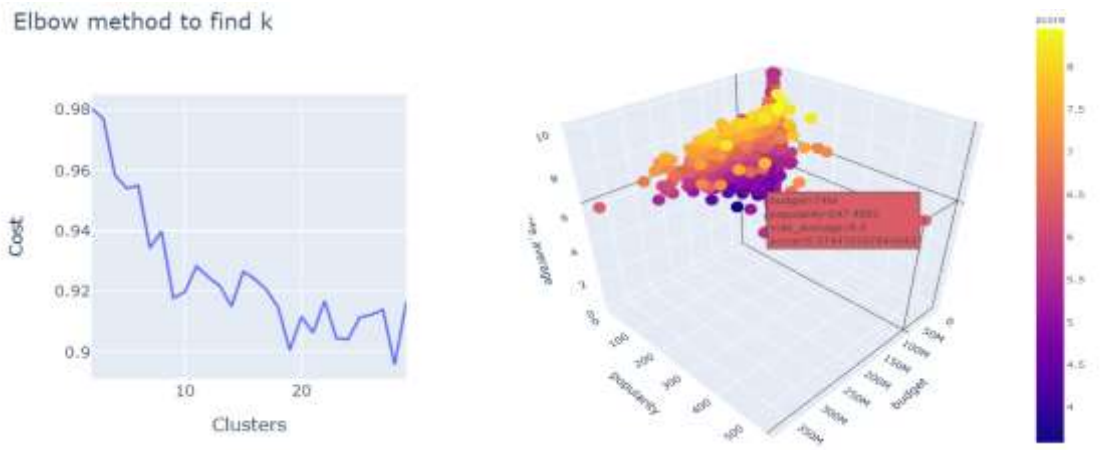


Figure 7.1: Elbow method to find the optimal number of clusters (left). Cluster of data point, each color represents a cluster. (right)

	adult	budget	popularity	revenue	vote_average	vote_count	score
23742	False	3300000	64.3	13092000.0	8.3	4376.0	8.205405
586	False	19000000	4.30722	272742922.0	8.1	4549.0	8.015676
1632	False	10000000	15.0648	225933435.0	7.9	2880.0	7.779907
891	False	878000	13.9161	10462500.0	7.9	1462.0	7.674918
582	False	100000000	22.6617	520000000.0	7.7	4274.0	7.624880

Figure 7.2: K-Means recommendation for a movie in cluster 8

	id	adult	budget	popularity	revenue	vote_average	vote_count	score
581	812.0	False	28000000	16.3574	504050219.0	7.4	3495.0	7.322002
619	1592.0	False	30000000	10.3262	102616183.0	7.4	644.0	7.045419
608	10112.0	False	4000000	9.83405	55675257.0	7.1	1287.0	6.936156
35	687.0	False	11000000	6.89132	39363635.0	7.3	350.0	6.772387
787	1645.0	False	40000000	13.309	152266007.0	7.0	522.0	6.675832

Figure 7.3: K-Means recommendation for a movie in cluster 17

References

[1] https://www.kaggle.com/rounakbanik/the-movies-datast

[2] https://grouplens.org/datasets/movielens/latest/

[3] https://realpython.com/build-recommendation-engine-collaborative-filtering/

[4] https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401

[5] https://arxiv.org/abs/1708.05031

[6] https://towardsdatascience.com/paper-review-neural-collaborative-filtering-explanation-implementation-ea3e031b7f96

[7] https://arxiv.org/pdf/1802.05814v1.pdf

[8] https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch

[9] https://rsandstroem.github.io/sparkkmeans.html

[10] http://kvfrans.com/variational-autoencoders-explained/

GitHub Repo:
https://github.com/waghmareomkar/Reccomendation_System