Performance analysis of different Recommendation Systems on the Movie Review Dataset

Hiral Nagda
Rutgers University
hiral.nagda@rutgers.edu

Yash Nisar
Rutgers University
yash.nisar@rutgers.edu

Abstract

In this work we provide a review of the experiments we conducted on two contrasting recommender systems' algorithms: classic Collaborative Filtering and Item-based Filtering. We discuss the results extracted from the experiments and test the performance of each of the different recommendation systems. Finally, the results are utilized to evaluate the methods' usefulness.

Introduction

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

The most common types of recommendation systems are **content based** and **collaborative filtering** recommender systems. In collaborative filtering the behavior of a group of users is used to make recommendations to other users. Recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie.There are two types of collaborative models **Memory-based** methods and **Model-based** methods.The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

- **User-based collaborative filtering**: In this model products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example if Derrick and Dennis like the same movies and a new movie comes out that Derick likes,then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- **Item-based collaborative filtering**: These systems identify similar items based on users' previous ratings. For example if users A,B and C gave a 5 star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A,B and C.

Problem Formalization & Data Set

We intend to analyze the performance of 3 major recommendation systems, i.e. Singular Value Decomposition (SVD), K-Nearest Neighbours, Slope-One. Each of them views the recommendation task from a different perspective and it will be interesting to see how these perspectives are reflected in the results of our experiments.

In order to execute our experiments we used the original MovieLens 1M data set. The data set consists of 1 million ratings from 6000 users on 4000 movies. All ratings follow the 1 (bad) - 5 (excellent) numerical scale. Finally, we have to note that the initial data set was used as the basis to generate five distinct splits into training and test data. To split the dataset we considered each user, randomly selected 80% of his/her ratings as the training ratings, and used the remaining 20% ratings as testing ratings.

```
In [27]: ratings = pd.read_csv('/Users/yashnisar/data/ratings.csv', sep=',', usecols=['userId', 'movieId', 'rating','timestamp']
         ratings.head()
```

Out[27]:

	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

Exploratory Data Analysis on MovieLens 1M data set.
The number of unique users in the data set was 610.

```
In [46]: # Number of unique users
         ratings['userId'].nunique()

Out[46]: 610
```

Number of unique movies in dataset was found to be close to 10000.

```
In [47]: # Number of unique movies
         ratings['movieId'].nunique()

Out[47]: 9724
```

Movie IDs of top 10 most rated movies. The movie with MovieID 356 was the most rated movie with 329 ratings.

Top 10 most rated movies

```
In [48]: most_rated = ratings.movieId.value_counts()[:10]
         print('MovieID\tNo.of ratings')
         most_rated

         MovieID No.of ratings

Out[48]: 356     329
         318     317
         296     307
         593     279
         2571    278
         260     251
         480     238
         110     237
         589     224
         527     220
         Name: movieId, dtype: int64
```

Aggregation of the total number of ratings and the average of all ratings

In [49]:
```
stat = ratings.groupby('movieId').agg({'rating': [np.size, np.mean]})
stat.head()
```

Out[49]:

	rating	
	size	mean
movieId		
1	215.0	3.920930
2	110.0	3.431818
3	52.0	3.259615
4	7.0	2.357143
5	49.0	3.071429

Taking only those movies into consideration that have been rated by at least 50 times

In [51]:
```
big_sample = stat['rating']['size'] >= 50
stat[big_sample].sort_values([('rating', 'mean')], ascending=False)[:10]
```

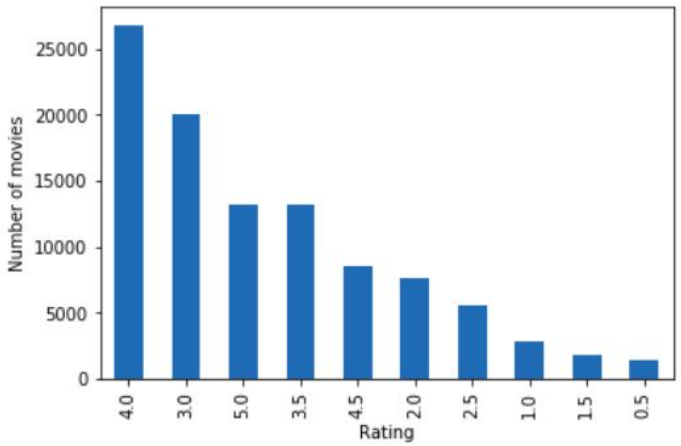
Out[51]:

	rating	
	size	mean
movieId		
318	317.0	4.429022
858	192.0	4.289062
2959	218.0	4.272936
1276	57.0	4.271930
750	97.0	4.268041
904	84.0	4.261905
1221	129.0	4.259690
48516	107.0	4.252336
1213	126.0	4.250000
912	100.0	4.240000

On visualizing the count of different ratings, we found that rating 4 was rated the most number of times and rating 0.5 was rated the least number of times.

```
In [52]: ax = ratings['rating'].value_counts().plot(kind='bar')
         ax.set_xlabel("Rating")
         ax.set_ylabel("Number of movies")
```

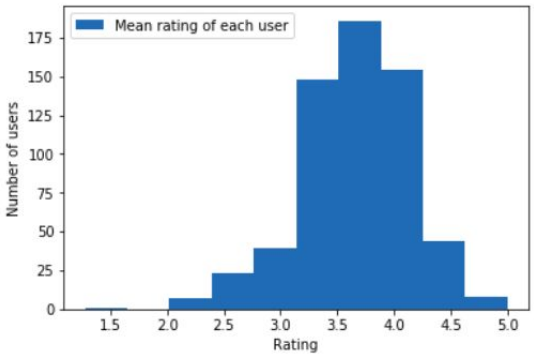
```
Out[52]: Text(0, 0.5, 'Number of movies')
```



We gathered all movies rated by a single user and calculated the mean rating given by each user. The given histogram shows that majority of the users have given a rating between 3 and 4. Also, we verified that the total number of users (given by summing height of the histogram) to be 610.

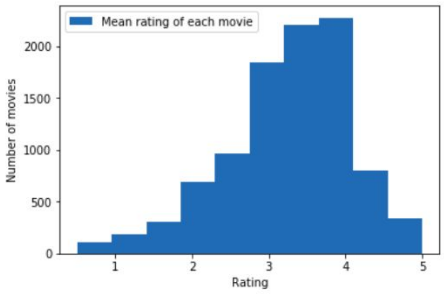
```
In [59]: ax = ratings.groupby('userId').agg({'rating': [np.mean]}).plot(kind='hist')
         ax.set_ylabel('Number of users')
         ax.set_xlabel('Rating')
         ax.legend(['Mean rating given by each user'])
```

```
Out[59]: <matplotlib.legend.Legend at 0x123ef8dd8>
```



We gathered all users who rated a particular movie and calculated the mean rating of each movie. Again, most of the movies are given a rating between 2.5 and 4. Also, we verified that the total number of movies (given by summing height of the histogram) to be 9724.

```
In [60]: ax = ratings.groupby('movieId').agg({'rating': [np.mean]}).plot(kind='hist')
         ax.set_ylabel('Number of movies')
         ax.set_xlabel('Rating')
         ax.legend(['Mean rating of each movie'])
Out[60]: <matplotlib.legend.Legend at 0x123f7b898>
```

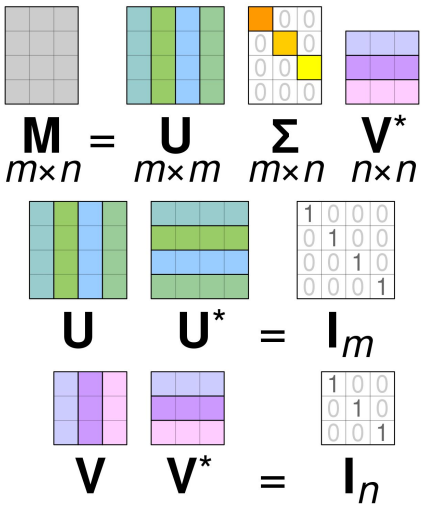


Proposed Model & Experiments

We've compared evaluation metrics for 3 main algorithms as follows:

```
for index, algo in enumerate([SVD(), SlopeOne(), KNNBasic(), KNNWithMeans()]):
    print("\n Algorithm:", algo_names[index])
    algo.fit(trainset)
    predictions = []
    for _, row in test_set.iterrows():
        predictions.append(algo.predict(row.userId, row.movieId, row.rating))
    print("RMSE", accuracy.rmse(predictions, verbose=False))
    print("MAE", accuracy.mae(predictions, verbose=False))
    # print(predictions)
    top_recommendations = recommend_items(predictions)
    norm_dcg = norm_dcg_score(predictions, k=10, gains="exponential")
    precisions, recalls = precision_recall_at_k(predictions, k=10, threshold=4)
    avg_norm_dcg = sum(ndcg for ndcg in norm_dcg.values()) / len(norm_dcg)
    avg_prec = sum(prec for prec in precisions.values()) / len(precisions)
    avg_recall = sum(rec for rec in recalls.values()) / len(recalls)
    fscore = 2 * (avg_prec * avg_recall)/ (avg_prec + avg_recall)
    print("Fscore:", fscore)
    print("Average Precision:", avg_prec)
    print("Average Recall:", avg_recall)
    print("Average Normalized Discounted Cumulative Gain:", avg_norm_dcg)
    # print("Top recommendations: ", top_recommendations)
```

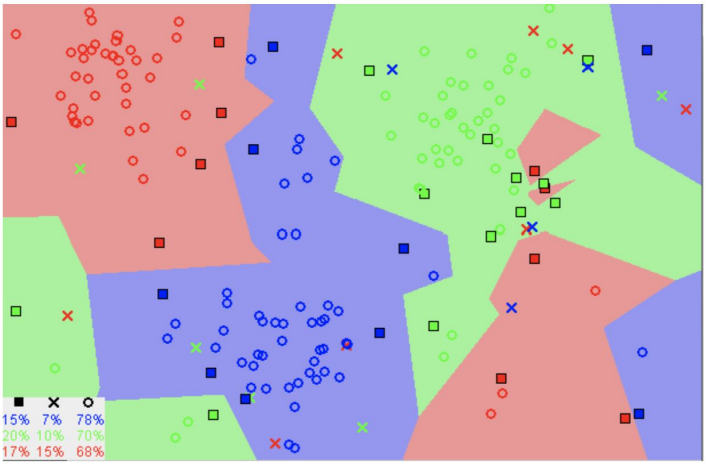
Singular Value Decomposition: In linear algebra, the singular-value decomposition is a factorization of a real or complex matrix. It is the generalization of the eigendecomposition of a positive semidefinite normal matrix to any matrix via an extension of the polar decomposition. It has many useful applications in signal processing and statistics. Visualization of the matrix multiplications in SVD are as follows:



The model evaluation metrics for the SVD algorithm are:

```
 Algorithm: SVD
RMSE 0.8738800226372638
MAE 0.6728995332406251
Fscore: 0.4527493365975382
Average Precision: 0.8428168097840228
Average Recall: 0.30950573121394714
Average Normalized Discounted Cumulative Gain: 0.8967618485224594
```

K-Nearest Neighbours: The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space.



The model evaluation metrics for the K-Nearest Neighbour algorithm are:

```
 Algorithm: KNN Basic
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE 0.9426463284177825
MAE 0.7252180060050526
Fscore: 0.4631585846481646
Average Precision: 0.7811403851157946
Average Recall: 0.32916438220170313
Average Normalized Discounted Cumulative Gain: 0.8951471374795963

 Algorithm: KNN with Means
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE 0.8965142287212152
MAE 0.6860245965663925
Fscore: 0.4818958691422948
Average Precision: 0.8174707259953162
Average Recall: 0.34164804224718215
Average Normalized Discounted Cumulative Gain: 0.8838221901993725
```

Slope-One: To drastically reduce overfitting, improve performance and ease implementation, the Slope One family of easily implemented Item-based Rating-Based collaborative filtering algorithms was proposed. Essentially, instead of using linear regression from one item's ratings to another item's ratings (f(x)=ax+b), it uses a simpler form of regression with a single free parameter (f(x)=x+b). The free parameter is then simply the average difference between the two items' ratings.

The model evaluation metrics for the Slope-One algorithm are:

```
 Algorithm: SlopeOne
RMSE 0.8994080546710016
MAE 0.6891764370426648
Fscore: 0.4661738057286894
Average Precision: 0.7906759042414774
Average Recall: 0.33052337336987747
Average Normalized Discounted Cumulative Gain: 0.8859103402500115
```

Item Recommendation

We created a recommendation list for a user is to predict the ratings on all the items that user didn't buy before, then ranked the items in descending order of the predicted rating. Then we filtered the top 10 items from the recommendation list.
We create a dictionary where the key is the userID, and the value is the list of 10 tuples in the decreasing order of their scores. These are the top 10 items from the recommendation list.

```
def recommend_items(predictions):
    results = {}
    for uid, iid, true_r, est, details in predictions:
        if uid in results:
            results[uid].append((iid, est))
        else:
            results[uid] = [(iid, est)]

    for uid in results:
        results[uid].sort(key=lambda x: x[1], reverse=True)
        results[uid] = results[uid][:10]

    return results
```

The output is as follows:
{1: [(608, 4.9011004992140395), (1967, 4.86977091687895), (1089, 4.860467371798719), (1265, 4.812860549110109), (1222, 4.766872539430366), (3578, 4.711305493358913), (1219, 4.697322302699699), (1206, 4.66059182562638), (2078, 4.660263792843132), (2947, 4.560454122988887)],
2: [(48516, 4.470894974121505), (58559, 4.302013414008627), (79132, 4.188958078905497), (89774, 3.6407958960283655), (114060, 3.6026737843102827), (86345, 3.4667376333018645)],
……}

From the given output, we deduce that the user with userID 1 will most probably watch the movies with movieID 608, 1967, 1089, 1265, 1222, 3578, 1219, 1206, 2078, 2947 in the decreasing order of priority and so on.

Conclusion and Future work

Evaluation metrics for different algorithms, which are calculated based on our test data (20% holdout data), are summarised in a tabular format as follows:

	SVD	Slope-one Algorithm	K-Nearest Neighbours (Basic)	K-Nearest Neighbours (with means)
RMSE	0.8738	0.8994	0.9426	0.8965
MAE	0.6728	0.6891	0.7252	0.6860
F-score	0.4527	0.4661	0.4631	0.4818
Precision	0.8428	0.7906	0.7811	0.8174
Recall	0.3095	0.3305	0.3291	0.3416
NDCG	0.8967	0.8859	0.8951	0.8838

We observe that SVD has the lowest RMSE and MAE value. Next higher values are for K-Nearest Neighbours (with means). K-Nearest Neighbours(Basic) has the highest RMSE and MAE value. The F-score values are almost similar for all the 4 models. Precision for SVD is highest and K-Nearest Neighbours (Basic) has the lowest precision. Recall for K-Nearest Neighbours (with means) is the highest and for SVD it is the lowest. F-score is the average of Precision and Recall when they are close. F-score for all the models are similar, K-Nearest Neighbours (with means) has the highest F-score value. Hence, we can conclude that SVD is a better model than Slope-one and K-Nearest Neighbours(Basic) and K-Nearest Neighbours(with Means) as it has the lowest MAE and RMSE value.

Acknowledgement

This research was supported by Rutgers University. We would further like to thank Dr. Yonfeng Zhang whose insights and expertise greatly assisted the research and make the experience worthwhile.