Performance Comparison of Algorithms for Movie Rating Prediction

ABSTRACT

Recommender systems are really critical in some industries like the entertainment and eCommerce industry as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. Lately, the demand for recommendation services has severely increased due to the massive flow of new content on the internet. It is crucial that services recommend the correct items, as it leads to increased consumption, increased user satisfaction, increased profit, and is beneficial to everyone. Recommendation based algorithms are used in a vast amount of websites, such as the movie recommendation algorithm on Netflix, the music recommendations on Spotify, video recommendations on Youtube, and the product recommendations on Amazon. As proof of the importance of recommender systems, a few years ago, Netflix organized a challenge called the "Netflix prize" with an award of one million dollars to whoever could implement the best movie recommendation software for them to use on their service. In this project, we will be working on the dataset collected by the GroupLens Research Lab at the University of Minnesota. The goal of this project is to compare different models by training them on the given inputs to predict movie ratings in the validation set. We evaluate these models using RMSE which will show us how close the predictions are to the true values hence resulting in better recommendations for users.

DATASET

We have used the smaller version of MovieLens latest dataset available on the website of the GroupLens Research Lab at the University of Minnesota. The dataset has about 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. We have the following variables:

- Quantitative features(discrete) - userId, movieId, timestamp
- Qualitative features(nominal) - title, genres
- Outcome(continuous) - rating

REGRESSION MODELS

We used regression models as a recommendation model which can be considered as one of the simplest ways to build a recommendation engine. The simplest model for this case would be to consider the same rating for all the movies and users and explaining all the variations and differences by the randomness factor or the error term. Following are models we used with more advanced approach:

1. **Movie effect**: We know that all movies have different reviews, and some are rated higher than the others, we can extend our simpler model by adding an effect which is movie specific, which can be

considered as bias in the model. This bias factor helped us to represent an average factor for a movie and following is the way we can show this model mathematically,

$$Yu,i = \mu + Bm + \varepsilon u,i$$

2. Movie and User effects: We also consider the user effect in this model. That is, some users are more active than the others so to get a more precise outcome, we added this term as an improvement. Mathematically,

$$Yu,i = \mu + Bm + Bu + \varepsilon u,i$$

3. Movie, User, and time effects: From the data exploration we noticed that apart from the rating and the activity status of the user, it is also important the time when (s)he rated the movie, it is a smooth function of both previous actors we discussed in the two models, so we added that function along with those 2 factors, the term is known as time effect.

$$Yu,i = \mu + Bm + Bu + f(du,i) + \varepsilon u,i$$

Where, **μ** is the true rating of a movie, **Bm** is the bias effect or movie specific effect, **Bu** is known as user-specific effect, **εu,i** is the error term which is independent and centered around 0, **du,i** is a DateTime object we got by rounding up to weekly units from converting the timestamp value f is a smooth function of **du,i**

4. Regularization: Regularization of the model permits us to penalize large estimates that come from small sample sizes. It is similar to the Bayesian approach which is also useful to shrink our predictions. The idea of this is to add a penalty for large values of user and movie specific bias by using the sum of squares method to minimize the values. But having many larger values for both makes it harder to minimize. By solving the least squares problem,

$$1/N * \sum u,i(yi,u-\mu-bm-bu)^2 + \lambda(\sum bm^2 + \sum bu^2)$$

the first term, i.e. the summation of the errors strives to find both movie and user effects that fits the given rating. The second term, which is the regularizing term helps us to avoid overfitting by adding the lambda, which penalizes the magnitude of the parameters. To pick the best lambda, we used cross validation, and using that value we came up with the minimum RMSE value.

Methods	RMSE
movie effect	**0.9748393**
movie + user effects	**0.8926160**
movie + user + time effects	**0.8913433**
Regularized Movie + User Effect	**0.8641621**

RECOMMENDATION MODELS:

There are basically three important types of recommendation engines:

- Collaborative filtering.
- Content-Based Filtering.
- Hybrid Recommendation Systems.

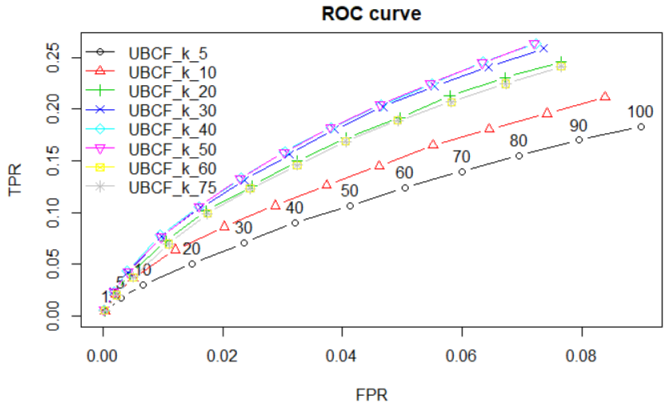
Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. Collaborative filtering based models can be broadly divided into Memory-Based Collaborative Filtering and Model-Based Collaborative filtering. We have only focused on the Memory-Based Collaborative Filtering technique models. There are 2 main types of memory-based collaborative filtering algorithms:

User-User Collaborative Filtering(UBCF): Recommendations based on user-user similarity

Item-Item Collaborative Filtering(IBCF): Recommendations based on item-item similarity

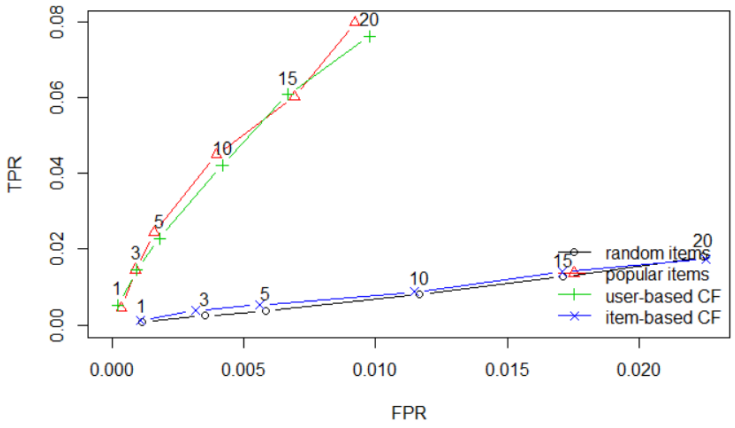
In either scenario, we need a similarity matrix. For both the models, we used cosine similarity.

We have used the recommenderlab package in r to build the models. We have done parameter tuning of both the models by optimizing precision and recall. We used 10-fold cross-validation, rating_threshold as 4 (threshold with the minimum rating that is considered good to recommend), and a number of items to generate recommendation as 10 for parameter tuning. We have built and evaluated the models over a range of values of n(number of recommendations). We have found the UBCF model with parameter nn=50 and IBCF model with parameter k=15 as the best models based on precision vs recall and TPF vs FPR graphs. Below find the obtained graph of ROC curves while parameter tuning UBCF models.



Further, we have compared the optimized UBCF and IBCF models along with popular items based model (Recommendations based on item popularity) and random items based model (Produces random recommendations). Both popular and random based models are basic and simple models. Recommendation models can be built for two outputs, one is predicting ratings and another is directly recommending top n items. So we have built the models both ways and compared using RMSE in the first case and ROC curves in the second case. Below find the results. We can observe that UBCF and popular item recommendation models are best in both cases but UBCF is personalized hence has an advantage over the popular item model.

methods	rmse	mse	mae
Popularity Model	0.8443124	0.7128635	0.6440923
Random Model	1.2276260	1.5070657	0.9546300
UBCF Model	0.8998878	0.8097980	0.6831003
IBCF Model	1.2114492	1.4676091	0.9312562



TREE-BASED MODELS

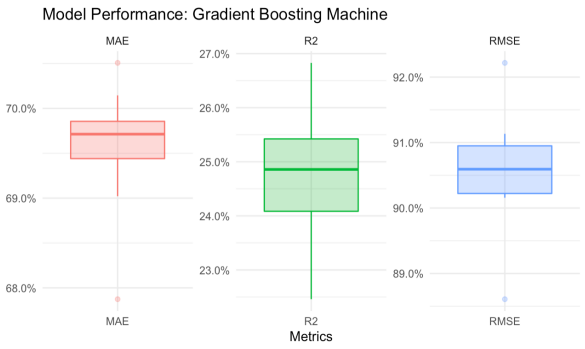
We used the h2o package in R to build and test the following models:

1. Gradient Boosting Machine(GBM)
2. Random Forest models(RF)
3. Stacked Ensemble

H2O provides many parameters for tuning in order to optimize our model. The hyperparameters that we have considered include ntrees, max_depth, learn_rate, and nfolds. We first built both the models using default values of ntrees, max_depth, and learn_rate which are 50, 5, and 0.1 respectively with k for cross-validation equal to 5. In the first model, we incorporate automated stopping so that we can increase the number of trees to 1000 but terminate training after 10 consecutive trees have no improvement on the cross-validated error. In the second case, we simply increased the number of folds for cross-validation to 10 keeping all other parameters at the same value as in the first model. For the third model, reduced the learn_rate to 0.05 in GBM and increased the max_depth to 20 in RF. We find that the second model of GBM and third model of RF have the lowest rmse for the same parameters, so we stacked these two models together

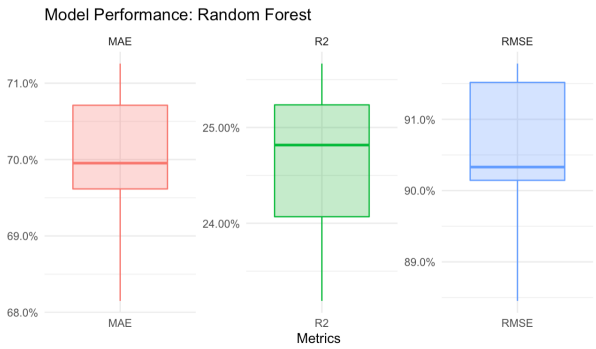
The following table compares the performance of GBM for the different cases:

Models	RMSE	MAE	R2
Default	0.9087178	0.7001256	0.2418579
Model 1	0.9090786	0.6981826	0.2412558
Model2	0.9051475	0.6948109	0.2478036
Model3	0.9088122	0.7012451	0.2417004



The following table compares the performance of RF for the different cases:

Models	RMSE	MAE	R2
Default	0.9055927	0.6967001	0.2470635
Model 1	0.9109043	0.7005048	0.2382052
Model 2	0.9085031	0.6985231	0.2422161
Model 3	0.9059894	0.7001367	0.2464036



Methods	RMSE
gradient Boosting	0.9087971
random forest	0.9213899
stacked ensemble	0.9028550

CONCLUSION

In this project, we have examined the potential best recommender system algorithm to predict movie ratings for the 100k version of the Movielens latest data. Using the provided training set and validation set, we successively trained different linear regression models, recommender engines and tree based ensemble methods. The model performances were evaluated through the RMSE (root mean squared error) showed that among the linear regression models, the model with regularized effects on users and movies is efficient, among the recommender systems UBCF is an appropriate model and among the tree based methods, stacked ensemble gave the best performance. Overall, linear regression model with regularized effects on users and movies gave the best performance with the lowest RMSE.

There are many other methods which have the potential to give better results like Matrix Factorization and Slope One methods. We can also try to further tune the tree based methods with appropriate parameters.