Professor: Dr. Huzefa Rangwala
CS 584-HW 4 Nov 9, 2021

Recommender System- Movies

REQUIRED INFORMATION

Mason User Id: jbidhan, skakarap Miner2 Usernames: flash, TheAverageMiner
Rank and RMSE Score: 297 and 0.81

DESCRIPTION OF APPROACH:

We approached the problem reading the question and making notes for the steps required. The approach we used can be summarized as below:

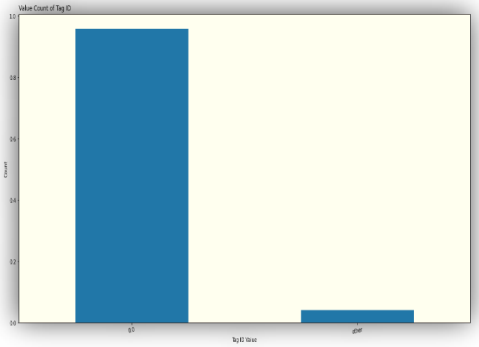
1. **Reading and storing the data.** Using pandas.read_csv : We created a method (function:readfile()) which needed a parameter for filepath to read the input. Using pandas.read_csv() function read the file and stored as a pandas' DataFrame. We stored all the dataframes from various .dat files. The test and train files had columns separated by " " and movie_actors, movie_directors, movie_genres, movie_tags, tags and user_taggedmovies are separated by "\t".
2. **Analysis of data:** We then analysed each column to understand the data inside different dataframes. We found how to make a roboust dataset out of them by merging them. We could merge train data with user_taggedmovies on ['userID','movieID'], movie_tags on ['movieID','tagID'], tags on 'tagID' and movie_genres, movie_directors, movie_actors on 'movieID'. The changed id column in tag to tagID to make it 'tagID' to maintain uniformity before merging. Similarly, we could merge it for test. We saw that there was a lot of categorical data like director's name, directors id, actor name, genre etc.

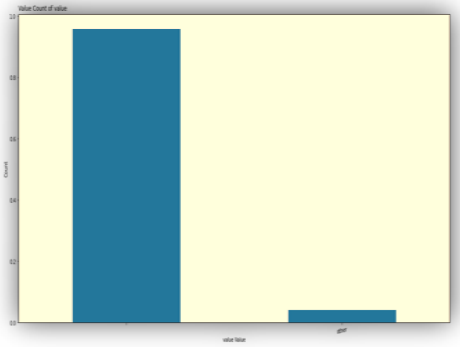
	userID	movieID	rating	tagID	tagWeight	genre	directorID	directorName	actorID	actorName	ranking	value
0	75	3	1.0	NaN	NaN	Comedy	donald_petrie	Donald Petrie	annmargret	Ann-Margret	3.0	NaN
1	75	3	1.0	NaN	NaN	Comedy	donald_petrie	Donald Petrie	buck_henry	Buck Henry	8.0	NaN
2	75	3	1.0	NaN	NaN	Comedy	donald_petrie	Donald Petrie	buffy_sedlachek	Buffy Sedlachek	13.0	NaN
3	75	3	1.0	NaN	NaN	Comedy	donald_petrie	Donald Petrie	burgess_meredith	Burgess Meredith	4.0	NaN
4	75	3	1.0	NaN	NaN	Comedy	donald_petrie	Donald Petrie	christopher_mcdonald	Christopher McDonald	9.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
47651627	71534	62049	4.5	NaN	NaN	Sci-Fi	michael_radford	Michael Radford	roger_lloyd_pack	Roger Lloyd Pack	17.0	NaN
47651628	71534	62049	4.5	NaN	NaN	Sci-Fi	michael_radford	Michael Radford	rolf_saxon	Rolf Saxon	25.0	NaN
47651629	71534	62049	4.5	NaN	NaN	Sci-Fi	michael_radford	Michael Radford	rupert_baderman	Rupert Baderman	8.0	NaN
47651630	71534	62049	4.5	NaN	NaN	Sci-Fi	michael_radford	Michael Radford	shirley_stelfox	Shirley Stelfox	19.0	NaN
47651631	71534	62049	4.5	NaN	NaN	Sci-Fi	michael_radford	Michael Radford	suzanna_hamilton	Suzanna Hamilton	3.0	NaN

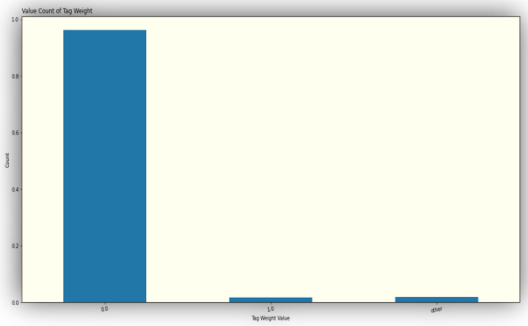
47651632 rows × 12 columns

We saw that there were many columns having NaN or no values. To Analyze it further, we went and checked for the value_counts and uniqueness in the columns.

- **TAG ID, TAG WEIGHT AND VALUES:** There were more than 80% of data is either null or empty that implies that not all movies have been reviewed by the user.



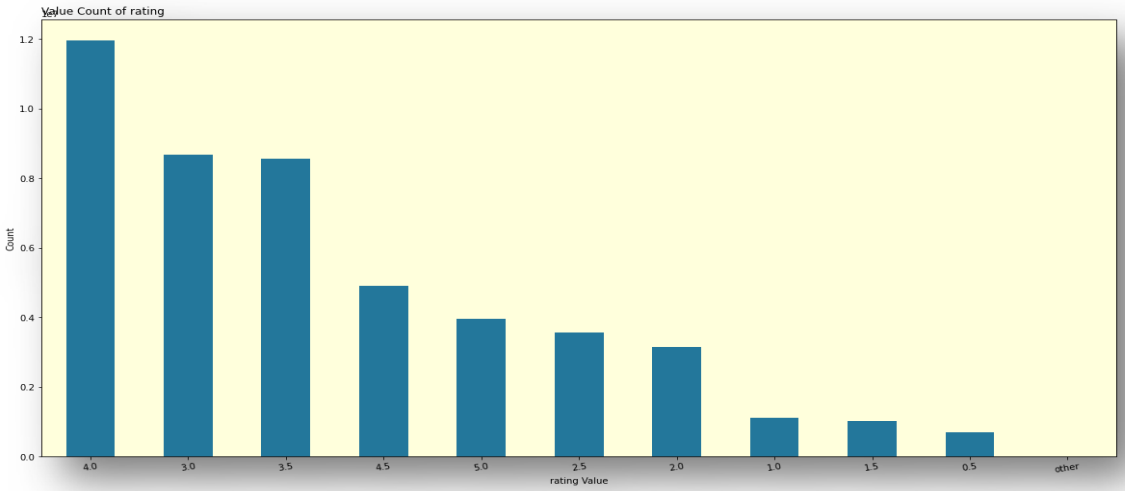




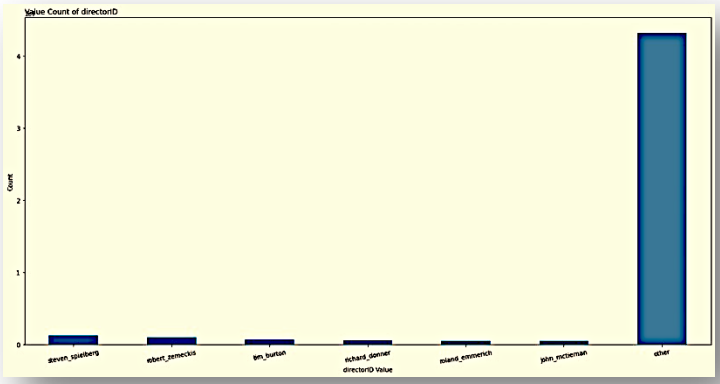
Professor: Dr. Huzefa Rangwala
CS 584-HW 4 Nov 9, 2021

- **RATING**: After merging all the tables on the train data, we found to have the following stat line for the rating column.

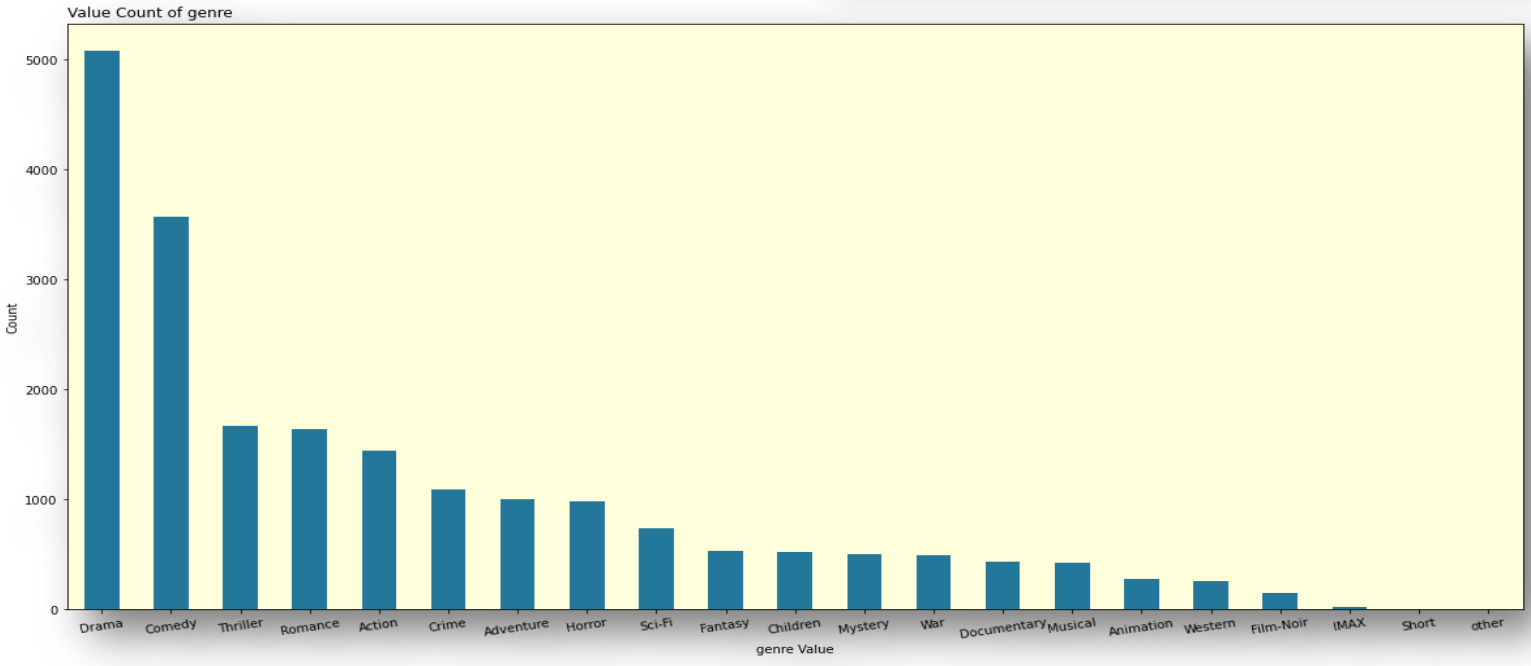
VALUE	COUNT
4.0	11970603
3.0	8689840
3.5	8571539
4.5	4913953
5.0	3964673
2.5	3571278
2.0	3141529
1.0	1107163
1.5	1026193
0.5	694861



- **DIRECTORID** is uniformly distributed and have some of the directos like "Steven Spielberg" directing most movies.



- **GENRE:** distribution over the data is shown below, most movies have Drama genre.



Janit Bidhan: jbidhan@gmu.edu Sashank Kakaraparty: skakarap@gmu.edu

3. **Preprocessing of the data**: We have processed the data in several different approaches and will include that in the methodology section.
4. **Experimentation: Each of these approaches is detailed in the methodology section.**
    a. **User-Based/Item-Based Collaborative Filtering**
    b. **Adding the Genre**
    c. **Adding the Genre + Directors**
    d. **Adding Genre + Directors + Actors**
    e. **Adding Tag as a part of the data**
    f. **Adding other additional data**
    g. **Using nothing but the Users + Movies + Ratings**
5. **Saving the output:** Then using the saveOutput() store the output in the .txt file and upload on the miner portal to get the results and then compared which of the following techniques and methods fared well.

*The library modules which were used were wrapped for modularity and the sake of reusability.

Methodology of Choosing the Approach and Associated Parameters and Graphs

We started writing the code on Jupyter notebook. We created two notebooks, one for analysis and the other with clean methods for the final submission. There were several factors involved in choosing the approach which are answered in the following points:

Selection of datasets to merge:

i. **User-Based/Item-Based Collaborative Filtering:** The first approach we worked on was to build a USERxMovie matrix with Users on the Y-Axis and the Movies on the X-axis and vice versa with MoviexUser. We tried the K-Nearest Neighbours model on the matrix but did never achieve an RMSE less than 1. Since the matrix was sparse, we applied a compressed sparse row matrix and KNN's resulted in a 0.2 dip in RMSE, an improvement but not substantial.
Note: We started to use additional data from hereon.

ii. **Adding the Genre:** We started with merging genre to the train data, as in general sense the genre seemed like a good encapsulation of the movie itself, which in turn might influence the user's impression of the movie. In analysis, we recognized that there are 20 unique genres in the dataset. However, since one movie could belong to multiple genres, we needed to find a way to convey the one-to-many relationship. We chose to one-hot-encode the genres resulting in a column representing each genre, with 1 standing for the movie's that fall under the given genre and 0 otherwise. We joined this genre data with the 'movieID' as key. We ran several regression algorithms on this data, more data on which is documented in the regression section.

iii. **Adding the Genre + Directors:** We continued to merge more data into our training, director ID coming in next. Director ID was an easier merge, considering each movie has only one director. So, we categorized all the directors from the 'movie_directors' file, to have a number representing it, which can be directly joined with the train data based on the 'movieID' column. We weren't entirely confident about its impact, because it's rare for a director to be an indication of the movie itself, unless it is Cristopher Nolan.

iv. **Adding Genre + Directors + Actors:** We continued down the same road and joined the actors into the mix. However, actors seemed to bring new challenges, starting with multiple actors being a part of the same film, which is different from the directors. We tried the same approach that we had with Genre's and one-hot-encoded the actors. However(again), unlike Genre's we had close to 6000 unique actors. One hot encoding and joining them with a 620K train data caused one of our computers to run out of memory. Finally, when we managed get the data merged, but ran into issues while running regression models, with our computers/cloud unable to run the entire matrix. We finally just dropped the experiment.

v. **Adding Tag as a part of the data:** We did try adding 'Tag' as a one-hot encoded column but realized it had a lot of sparseness because not all users were leaving tags on all the movies, which is also indicated in the graphs above.
vi. **Adding other additional data:** We debated merging the other additional data, however recognized that they have similar issues as actors i.e., many unique values.
vii. **Using nothing but the Users + Movies + Ratings:** This certainly must be the least expected outcome of this assignment and the most unintuitive approach we never would have thought of when we began. We achieved our best RMSE value when we had nothing but the basic columns. This, to our understanding goes against the basic premise that more information should be beneficial for the model, especially content-based information.
viii. **Rounding the rating to the nearest 0.5:** On our best performing set of values, we tried to round it off to the closest 0.5 to reduce the RMSE. This was just a hack and did not make it any better

Different Regression Algorithms:

i. **K-Nearest Neighbor:** We used KNN with K as 1500 to try and predict in all algorithms, our results were not very impressive with KNN, which is understandable considering it is a basic algorithm. The best RMSE we achieved with KNN was 0.98.
ii. **Random Forest Regressor:** We moved to Random Forest for all our models and got similar RMSE scores in the range of 0.96 – 1.01, which was marginally better than the KNN.
iii. **Extreme Gradient Boosting:** The next and the best performing model was decision tree ensembles using Gradient Boosting. The Extreme gradient boosting used as a regressor performed better than the random forest models, which are also ensembles of trees. Finalized on the parameters of the extreme gradient boosting regressor after iterating various possible options using searching within constraints and cross validation. The experimentation and fixing of the number of estimators (n_estimators), maximum tree depth (max_depth) and learning rate yielded in better performance of the models, while we went with some generally amicable values for sub-sample column and level ratios. The Extreme Gradient Boosting Regressor gave us a best Root Mean Square Error value of 0.81 on our final and best performing data.

CONCLUSION

On seeing the results of the various experiments on various additional attributes, applying various regressors on the data with additional parameters, Extreme Gradient Boosting Regressor gave the best Root Mean Square Error value of 0.81. The resulting rating predictions are placed in the outputs folder. The attached folder will also contain two notebooks, one of which is marked analysis and will contain all the experiments and attempts while the other is the work containing our best performing model.

REFERENCES:

Documentation from https://scikit-learn.org/ , https://www.nltk.org/, https://www.scipy.org , https://matplotlib.org/ , https://xgboost.readthedocs.io/en/latest/

Blogs**:** https://machinelearningmastery.com/xgboost-for-regression/, https://medium.com/geekculture/end-to-end-movie-recommendation-system-49b29a8b57ac