Recommendation System for Movies

Aria Pessianzadeh[*], Dheeraj Perumandla[*], Manognya Pendyala[*]

[*] Indiana University Bloomington

I. ABSTRACT:

In this project we are using Content-Based filtering: They recommend things depending on one. This method recommends movies based on genre, director, description, actors, etc. If a person enjoyed one item, they would also appreciate a comparable item. Dataset we are using this project is provided by MovieLens. We mainly work on recommendation part and prediction part. We are using similarity based and association filtering for recommendation and after comparison of models by pycaret library, we will be using tuned Gradient Boosting Regressor for prediction of 'vote_average'.

Keywords: Content-Based filtering, MovieLens, Recommendation system, Pycaret, Gradient Boosting Regressor.

II. INTRODUCTION

An entirely new epoch of information has emerged as data collecting has expanded at an unprecedented rate. Recommendation Systems are a prime example of how data is being utilized to improve the effectiveness of various infrastructures. Information filtering systems, of which Recommendation Systems are a subset, are tasked with enhancing the quality of search results by surfacing results that are more pertinent to the search item at hand or related to the user's previous queries.

They are employed to foretell how a user would rate or prefer an item. Almost every big digital business makes use of them in some way, from Amazon's product recommendations to YouTube's autoplay settings to Facebook's suggestions for sites to like and people to follow. Furthermore, the success and growth of businesses like Netflix and Spotify rest heavily on the quality of their recommendation engines[1].

III. RECOMMENDATION PART

In this part we work on developing the recommendation system. Coming to dataset, MovieLens users' 5-star ratings and free-text tagging behavior is detailed in this dataset (ml-latest-small). It covers 9742 films and has 100836 ratings plus 3683 tags. Between March 29, 1996, and September 24, 2018, 610 users contributed to these numbers. The date this dataset was produced was September 26th, 2018. Participants were chosen at random. Each of the chosen individuals has provided feedback on 20 or more films. There is no demographic breakdown provided. Only a unique identifier is provided for each user; no additional details are given.

Movie meta-data consists of variables such as 'adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id', 'imdb_id', 'original_language', 'original_title', 'overview', 'popularity', 'poster_path', 'production_companies', 'production_countries', 'release_date', 'revenue', 'runtime', 'spoken_languages', 'status', 'tagline', 'title', 'video', 'vote_average', 'vote_count', of which, 'homepage', 'status', 'video', 'original_title', 'imdb_id', 'poster_path', 'release_date', 'production_companies', 'production_countries', 'spoken_languages' and 'belongs_to_collection' are dropped as they are not required for the further analysis.

Ratings.csv is another dataset which constitutes data such as userId, movieId and rating. These are mapped with the metadata in the further steps of the analysis[2].

We then are considering movies which at least have 5 reviews in total as anything below that would not contribute to the analysis in any way.

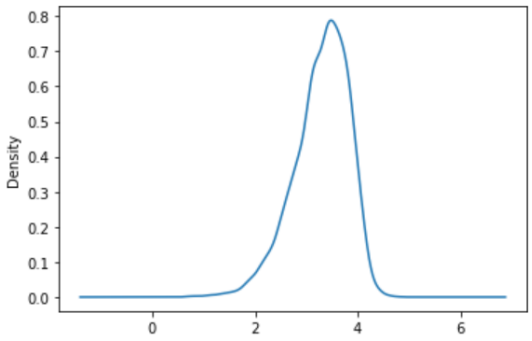

Fig 1: Distribution of ratings

The above figure 1, is the distribution of average movie ratings, we could observe that, most of the movies are rated around 3.5.

1. Recommendation based on Overview Similarity:

Overview is something like a storyline, it consists gist of the movie in one or 2 sentences.
In this step we are creating a recommendation model which recommends movies based on the similarity index it has with other movies. Using word to vector convert method and then using cosine similarity to compare the vectors of movies overviews. Movies that have overviews with more similar vectors, will be recommended together.

With the help of 'TfidfVectorizer()' module present in 'sklearn.feature_extraction.text', we can remove the stop words and tokenize the overview which are useful [3].

We are using cosine similarity to calculate the similarity among overviews [4].
It uses the formula cos(x, y) = x . y / ||x|| * ||y|| to get the cosine similarity.

We wrote a function recommend_movie(), which takes an already present movie as input and recommends top 5 movies which are similar to the input movie.

```
recommend_movie('Jumanji')
```

```
1                Jumanji
30398             Pixels
19726     Wreck-It Ralph
2486            eXistenZ
10892        Stay Alive
```

```
recommend_movie('Sense and Sensibility')
```

```
20                                  Get Shorty
9767                                   Be Cool
17696                  Waiting for Forever
10598                               King Kong
10554    Harry Potter and the Goblet of Fire
```

```
recommend_movie('Toy Story')
```

```
0                      Toy Story
15348                Toy Story 3
2997                 Toy Story 2
10301    The 40 Year Old Virgin
24523                  Small Fry
```

Fig 2: recommend_move() outputs

The above figure 2, are the examples of output of the recommend_movie() function.

2. Recommendation based on genres:

Later we are looking to recommend movies based on genres.

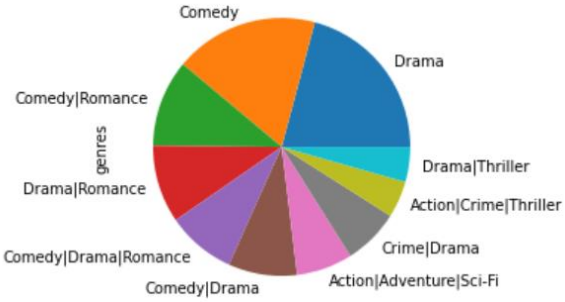

Fig 3: Genre distribution

The above chart, figure 3, describes the distribution of different genres across the whole dataset after merging the datasets.
The function best_movies_each_genre() returns top 10 movies in the input genre.

3. Recommendation based on associative rule:

It works like an association rule algorithm. Find the users that have watched a particular movie and then find the most common other movies that those users have watched! If a lot of users have watched movies a and b, and you have watched movie a, then you should probably watch movie b as well.

The function users_also_watched(movie) returns associated movies for the given input movie.

IV. PREDICTION OF RATING (VOTE AVERAGE)

Why is it necessary for us to make a guess on the rating here? Since a user's rating is one of the most significant factors to consider when developing a recommendation system for movies, we are utilizing various algorithms that are designed to anticipate ratings for films.

During this stage, pycaret will be utilized. PyCaret is a machine learning package written in Python that is open source and uses a low-code approach to automate machine learning workflows. It is a tool for managing models and doing machine learning from beginning to end, which dramatically increases your rate of productivity and speeds up the trial cycle[5].

The steps involved:

1. Pre-processing:

Here we are working extensively on movie meta-data data set.
The data set constitutes of features such as: 'adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id', 'imdb_id', 'original_language', 'original_title', 'overview', 'popularity', 'poster_path', 'production_companies', 'production_countries', 'release_date', 'revenue', 'runtime', 'spoken_languages', 'status', 'tagline', 'title', 'video', 'vote_average', 'vote_count'.

Out of these features, ‘revenue’, ‘runtime’, ‘vote_average’, ‘vote_count’ are numerical datatype features.

From, the above features, we are only considering 'popularity', 'adult', 'release_date', 'video', 'revenue', 'runtime', 'vote_count', 'vote_average' for further analysis as most of the others are nominal features and are not useful for regression analysis.

When checked for null values and zero values, revenue had like 37729 / 45130 values which are zeros, so we thought not much information can be extracted from feature revenue, so we decided on dropping it.

And features having a smaller number of zero values, ‘popularity’, ‘runtime’, ‘vote_count’, ‘vote_average’ we decided on filling them with the median values. And there were just a few ‘nan’ values throughout, so we decided on dropping those rows

Overview | Alerts 4 | Reproduction

Dataset statistics

Number of variables	7
Number of observations	45130
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	36
Duplicate rows (%)	0.1%
Total size in memory	3.8 MiB
Average record size in memory	87.4 B

Variable types

Numeric	5
Boolean	2

Fig 4: Overview

The above figure 9, describes the overview of the selected features, after modifying the null and zero value rows, for further analysis. We could see that there are no missing cells, 36 duplicate values, 5 numeric variable types and 2 Boolean(True/False) variable types.

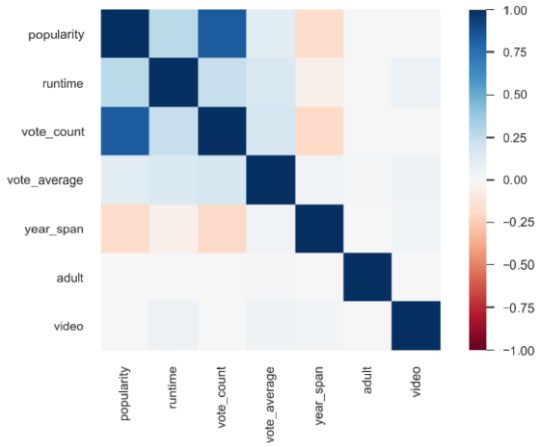

Fig 5: Correlation Heatmap

The above figure describes the correlation among the features. We could observe that vote_average doesn’t show high correlation with any of the features.

2. Model Setup:

We are using setup() module in pycaret.regression library to start the initialization of the process. This method sets the training and tests data in 70:30 ratio, performs one-hot encoding and sets fold generator to KFold, with fold number as 10 [6].

We now use compare_models() method to have a brief comparison among various machine learning regression algorithms.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.8292	1.2858	1.1338	0.1323	0.1859	0.1726	0.1370
gbr	Gradient Boosting Regressor	0.8369	1.2924	1.1367	0.1278	0.1867	0.1746	0.2480
ada	AdaBoost Regressor	0.8911	1.3978	1.1822	0.0566	0.1907	0.1794	0.0750
rf	Random Forest Regressor	0.8598	1.4023	1.1840	0.0536	0.1923	0.1771	0.6200
br	Bayesian Ridge	0.8923	1.4419	1.2007	0.0269	0.1960	0.1865	0.0290
lr	Linear Regression	0.8924	1.4422	1.2008	0.0267	0.1960	0.1865	0.5780
ridge	Ridge Regression	0.8924	1.4421	1.2008	0.0267	0.1960	0.1865	0.0340
lar	Least Angle Regression	0.8924	1.4422	1.2008	0.0267	0.1960	0.1865	0.0270
en	Elastic Net	0.8920	1.4429	1.2011	0.0262	0.1962	0.1866	0.0250
lasso	Lasso Regression	0.8922	1.4449	1.2019	0.0249	0.1963	0.1867	0.2430

Fig 6: Model Comparison

After comparing various models, see fig 6, the method highlights the best performing algorithms among all like shown in the figure. We have ‘Light Gradient Boosting Machine’, ‘Gradient Boosting Regressor’ and ‘AdaBoost Regressor’ as our top 3 best performing algorithms before hyperparameter tuning.

Now with tune_model() we tuned the top 3 models with it. After tuning, we have tuned Gradient Boost Regressor with learning_rate=0.05, max_depth=7, max_features=1.0, min_impurity_decrease=0.2, min_samples_leaf=5, min_samples_split=10, random_state=3487, subsample=0.85 as the best performing model.

We could still use untuned Light Gradient Boosting Machine, as it tops 2nd while comparing tuned and untuned models in combined. Untuned AdaBoost Regressor stands 3rd though. But AdaBoost is quite fast to train.

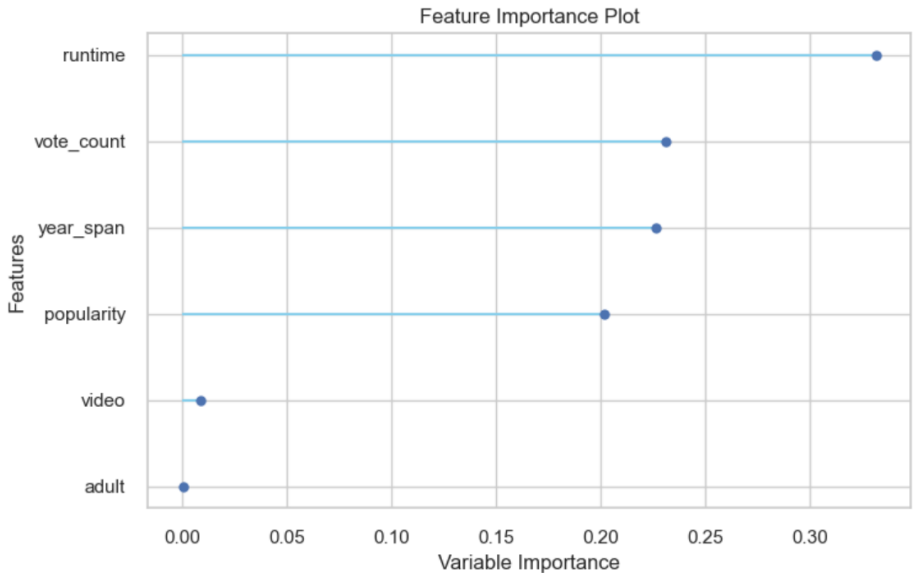

Fig 7: Feature Importance Plot

The above figure 7, plot is plotted to understand the importance of each variable. We could observe that runtime has the most importance, whereas 'adult' has the least importance on determining the 'vote_average'.

Now with the finalized regression model, we will test it on test data (The remaining 30% of data).

	popularity	adult	video	runtime	vote_count	year_span	vote_average	prediction_label
31590	20.720114	0.0	0.0	0.0	122.0	5.0	6.3	6.842580
31591	0.140197	0.0	0.0	100.0	1.0	32.0	8.0	5.464016
31592	0.814722	0.0	0.0	108.0	4.0	19.0	4.5	5.831901
31593	6.837575	0.0	0.0	0.0	107.0	8.0	4.9	6.561176
31594	2.388788	0.0	0.0	130.0	10.0	39.0	6.2	6.011562

Fig 8: Test Predictions

In the above figure, we could see that the 'prediction_label' (test results) are quite near to the 'vote_average' (actual values). The model has a MSE of 1.257 and R^2 of 0.12.

V. RESULTS

The Recommendation works as expected. We have created recommendation systems based on similarity, genre and association. And coming to prediction of vote_average, the trained model works with an MSE of 1.257 and R^2 of 0.12.

VI. REFERENCES

[1] https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system

[2] Yang, H. (2018). Data preprocessing. *Pennsylvania State University: Citeseer*.

[3] Huang, C. H., Yin, J., & Hou, F. (2011). A text similarity measurement combining word semantic information with TF-IDF method. *Jisuanji Xuebao(Chinese Journal of Computers)*, *34*(5), 856-864.

[4] Guo, G., Zhang, J., & Yorke-Smith, N. (2013, August). A novel Bayesian similarity measure for recommender systems. In *IJCAI* (Vol. 13, pp. 2619-2625).

[5] https://pycaret.gitbook.io/docs/

[6] Gain, U., & Hotti, V. (2021, February). Low-code AutoML-augmented data pipeline–a review and experiments. In *Journal of Physics: Conference Series* (Vol. 1828, No. 1, p. 012015). IOP Publishing.