Movie Recommendation System

Milestone 2 Report

Project Objective:

The primary objective of the project is to provide movie recommendations to the users based on their preferred movie input. The project is built on top of trustworthy and large datasets collected from 3 different sources. This truthfulness of the data, will help to provide real life movie recommendations.

Furthermore, this project will also focus on predicting the ratings for the movies using regression models.

Tool Type:

1. Recommendation system
2. Movies ratings prediction (secondary)

Data Used:

- IMDB movie datasets (by genre) (https://www.kaggle.com/datasets/rajugc/imdb-movies-dataset-based-on-genre)
- MovieLens Datasets (https://grouplens.org/datasets/movielens/)
- TMDB Dataset (https://www.kaggle.com/datasets/asaniczka/tmdb-movies-dataset-2023-930k-movies)

Tech Stack:

I have listed the tech stack which I used, along with the tech stack which I will be using for future work.

- Programming Language: Python
- Data storage: SQLite
- Data processing: Pandas, NumPy, Collections, sklearn
- Visualization: Matplotlib, Seaborn

Project Timeline:

- Milestone 1: Data Collection, Preprocessing, and Exploratory Data Analysis (EDA) Timeline: February 5, 2025 – February 23, 2025 (2 weeks)
  - Feb 5 – Feb 7: Identify and acquire datasets
  - Feb 8 – Feb 10: Verify dataset accessibility, licensing, and documentation
  - Feb 11 – Feb 15: Data preprocessing (handling missing data, outliers, scaling)

- ➢ Feb 16 – Feb 19: Exploratory Data Analysis (EDA) (statistical summaries, visualizations)
- ➢ Feb 20: Finalizing the EDA report and project documentation
- ➢ Feb 23: Submit Milestone 1 deliverables

❖ Milestone 2: Feature Engineering, Feature Selection, and Data Modeling Timeline: February 21, 2025 – March 21, 2025 (5 weeks)

- ➢ Feb 27 – Mar 3: Feature engineering (creating new features)
- ➢ Mar 5 – Mar 10: Feature selection
- ➢ Mar 11 – Mar 13: Splitting the dataset and initial model training
- ➢ Mar 15 – Mar 18: Model tuning and hyperparameter optimization
- ➢ Mar 20 – Mar 26: Model evaluation and comparison
- ➢ Mar 31 – Apr 2: Preparing Milestone 2 report
- ➢ Apr 7: Submit Milestone 2 deliverables

❖ Milestone 3: Evaluation, Interpretation, Tool Development, and Presentation Timeline: March 24, 2025 – April 23, 2025 (5 weeks)

- ➢ Apr 8 – Apr 10: Evaluate model performance on test data
- ➢ Apr 11 – Apr 12: Interpret model results and address biases
- ➢ Apr 13 – Apr 16: Develop an interactive dashboard for visualizations
- ➢ Apr 17 – Apr 18: Implement the final recommendation system
- ➢ Apr 19 – Apr 20: Tool testing and debugging
- ➢ Apr 21: Prepare final report and GitHub repository updates
- ➢ Apr 22: Record demo video and finalize presentation
- ➢ Apr 23: Submit Milestone 3 deliverables

EDA report:

In Milestone 1, the data was normalized. For further analysis, the same data has been stored in DB now. So that it can be fetched easily for next work.

In Milestone 2, for **Feature Engineering**. I performed aggregation on the data by applying complex joins and kept only the valid records in Dataset (46173 records). This clean data helped. Further performed data imputation.

```
(46173, 17)
/n ***** Converting Data for columns imdb_votes, year, release_year, budget, revenue from float to int *****
/n ***** Imputing Data for columns release_year, tmdb_vote_avergea *****
```

Further, performed imputations on columns like budget, revenue and votes to fill the gaps and to verify analysed the dataset

```
46063 14
Missing values per column:
  movieId: 0 missing (0.00%)
  movie_name: 0 missing (0.00%)
  runtime: 0 missing (0.00%)
  language: 0 missing (0.00%)
  popularity: 0 missing (0.00%)
  release_year: 0 missing (0.00%)
  budget: 0 missing (0.00%)
  revenue: 0 missing (0.00%)
  genres: 0 missing (0.00%)
  directors: 0 missing (0.00%)
  stars: 0 missing (0.00%)
  production_house: 0 missing (0.00%)
  rating: 0 missing (0.00%)
  votes: 0 missing (0.00%)
```

Now, features like genre, stars, directors and production_companies can be the key ones relating closely with movie_rating. But each of them had more than 10,000+ distinct entries. To keep it simple, instead of considering all, I only considered Top values for each attribute and performed TF-IDF for them.

Why TF-IDF? A particular move may have 5 stars in it, out of which 3 are very popular. If so, then each one gets equal weightage based on rating and other 2 also gets some kind of weightage (close to zero but non-zero). This made things simple, reduced the number of features and made the dataframe less sparse.

This are some (code generated) popular genres, stars, directors and production companies:

```
['Drama', 'Comedy', 'Romance', 'Action', 'Crime', 'Thriller', 'Horror', 'Adventure', 'Mystery', 'Fantasy']
['Nicolas_Cage', 'Bruce_Willis', 'Christopher_Lee', 'Gérard_Depardieu', 'Eric_Roberts', 'John_Wayne', 'James_
['Michael_Curtiz', 'Richard_Thorpe', 'Cheh_Chang', 'Alfred_Hitchcock', 'Mervyn_LeRoy']
['Unknown', 'Warner_Bros._Pictures', 'Metro-Goldwyn-Mayer', 'Paramount', '20th_Century_Fox']
```

Now, for **Feature Selection**, I performed some tests such as **Correlation Analysis** and **Tree-based feature selection**.

Correlation Analysis was performed on numerical columns like popularity, release_year, budget, revenue, votes, rating. And found below results:

	popularity	release_year	budget	revenue	votes	rating
popularity	NaN	0.09184	0.121412	0.133249	0.242904	0.104728
release_year	NaN	NaN	0.053815	0.007208	0.170181	0.137125
budget	NaN	NaN	NaN	0.302511	0.222280	0.099507
revenue	NaN	NaN	NaN	NaN	0.196978	0.067312
votes	NaN	NaN	NaN	NaN	NaN	0.403179
rating	NaN	NaN	NaN	NaN	NaN	NaN

Although the correlation between these variables is not strong enough. There is some level of correlation. Which indicates there is no linear correlation, but they are still useful.

Further, to find top_features, I used **Random Forest (Tree based Feature Importance)**. It ranks the features based on their importance on target (rating) in the dataset. Here are the results,

```
votes                          0.285324
runtime                        0.144640
popularity                     0.134923
release_year                   0.132341
genre_horror                   0.079131
genre_drama                    0.050674
budget                         0.031600
genre_action                   0.019502
genre_comedy                   0.017593
revenue                        0.016870
genre_thriller                 0.014850
genre_adventure                0.014430
genre_crime                    0.012214
prod_unknown                   0.010913
genre_fantasy                  0.009635
genre_romance                  0.009315
genre_mystery                  0.006844
prod_warner_bros._pictures     0.001702
prod_metro-goldwyn-mayer       0.001667
prod_20th_century_fox          0.001256
dtype: float64
```

The results says that genre has strong overall impact on ratings, along with some other attributes like number of votes, popularity, etc.

For **Data Modeling**, I planned to train 3 models. Linear Regression (to predict the movie rating based on trained data), Logistic Regression (to classify if a user will like the movie

based on trained data), KNN clustering (for unsupervised learning based recommendation).

For **Linear regression**, I removed the string type columns and split the data into ratios of 70:15:15 (70 % train, 15% validate, 15% test), and received the lowest MSE (Mean Squared Error). Which indicates that model performed quite well.

```
📊 Linear Regression Results
Validation MSE: 0.8367
Test MSE: 0.8312
```

For **Logistic Regression**, I classified the movie as good if it has rating >= 7. This classification was a bit imbalanced as the class split was 85:15.

```
Class distribution:
 is_good
0    38379
1     7684
Name: count, dtype: int64
```

So while training the model, I used the param “class_weight = ‘balanced’”. Again here I did a split as Training Data (70%), Validate Data (15%) and Test Data (15%). For performance evaluation, I checked Accuracy, F1 score and recall measurement for “good movie”

```
📊 Logistic Regression Results
Validation Accuracy: 0.7319
Validation F1 Score: 0.4959

Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.72      0.81      5832
           1       0.32      0.71      0.44      1078

    accuracy                           0.72      6910
   macro avg       0.62      0.72      0.63      6910
weighted avg       0.84      0.72      0.75      6910
```

Notice that when I classified data initially it had only 15% of good movies in total. So, our key analysis here is not just accuracy but to measure recall for “good movies” which is 72%. That makes a model quite balanced.

For **KNN classifier** (unsupervised model), we removed the string type variables again and trained the model on all features with k=6 (5 recommendations + 1 self). To verify the result, I checked the genre column data for all of the recommended movies (as genre was the most dominating one in feature selection above). They had similar genres.

	movie_name	genres
25184	Endgame - Bronx lotta finale	Action,Thriller,Sci-Fi
30766	Gunned Down	Action,Thriller
32964	Alterscape	Action,Thriller,Sci-Fi
36786	Disturbing the Peace	Action,Thriller
38238	The Last Sentinel	Action,Thriller,Sci-Fi
44620	Double Threat	Action,Thriller

Conclusion:

There is a slight accuracy difference between linear regression and logistic regression. However, I believe that it’s due to imbalanced data being used for logistic regression. Nonetheless, the difference is not much and at individual level the model satisfies the criteria as discussed above.

For KNN, we don’t really have a way to measure the accuracy in numbers as it's an unsupervised model. So, to measure its performance i checked cosine similarity and manual verification of the recommendations