Predicting IMDB Movie Ratings

Andy Gnias

December 14, 2024

1 Introduction

The ability to predict how a movie will rank among audiences before it is seen can benefit several stakeholders. If a video streaming service is deciding whether to host a new movie, audience favorability will hold significant value. One metric of audience ranking is a movie's rating on the Internet Movie Database, referred to as IMDB. IMDB utilizes user rankings to give movies a rating from 1 to 10 with 0.1 granularity. The goal of this project will be to create a model that can accurately predict the IMDB ranking of a movie.

Movie metadata data will be used to generate features. On Kaggle, user OctopusTeam has uploaded datasets containing movie data from streaming services Netflix, Apple TV, Amazon Prime, Hulu, and Max[3]. In total, this gives a dataset of around 67,000 movies before preprocessing. New features can also be added from other external sources. When adding features, features that could indicate a movie's success or failure after release should not be used. For example, gross revenue would be a good indicator of a movie's success, but in a real world example, if revenue can be found, critical reviews can also be found. The real value in the model described in this project would be the ability to predict a movie's ranking before an audience as seen and critiqued it.

This is a regression problem. Although it can be modeled as a classification problem, ex. above or below a certain rating, the goal of this project will be to see how close the model can get to predicting a movie's actual IMDB rating. In attempting to solve this problem, several different approaches will be tested: linear regression, ensemble methods, SVMs, and neural networks. Models for each approach will be developed, and results of each model will be compared.

2 Data Analysis

The dataset was initially seeded with 5 CSV files from Kaggle User OctopusTeam[3]. These CSVs contained movies and TV shows from major streaming services. The fields included Title, Type (movie or TV show), Genre, Release Year, IMDB ID, IMDB Rating, IMDB Number of Votes, and Available Countries. Title was used as an identifier, and anything without a title was dropped. Type was used to filter out TV shows. IMDB Rating was used as the label, and anything with a 0 was dropped, as that indicated that no rating was present. IMDB Number of Votes and Available Countries were disregarded, as they could indicate the movie's success after release. This left us with Genre and Release Year as features. Genre was a comma-separated list. This was one-hot encoded, where for each genre the movie was classified as, a 1 would be added to that category's column. Release year was normalized via Z-score and used as a feature, and any sample without a release year was dropped. Any duplicate movies were removed.

These features seemed insufficient for a robust model. However, with the IMDB ID, the IMDB page for each movie could be found. Initially, issuing a Python GET request to these pages gave a 403 error, likely due to the site's security policies. These were worked around by using a Python requests Session object that included a User-Agent header, making it seem more like a browser, and a rate-limiting mechanism to not overload the site or seem like bot traffic. Implementing these measures allowed the following features to be scraped from IMDB directly: Director, Runtime, Content Rating (ex. G, PG, PG-13, R), and Description. Runtime was converted to minutes and normalized via Z-score, and Content Rating was 1-hot encoded. From here, a Baseline dataset was defined with 60,940 samples and 62 features.

2 separate expansions to this dataset were generated. In one expansion, the directors of each movie were 1-hot encoded. However, this resulted in a dataset with 31,237 features, which made its use impractical for most models. Another expansion was to perform sentiment analysis via the Hugging Face Transformers library on the description of each movie, and utilize the sentiment score as a feature. For example, a movie with an extremely positive description would have a sentiment score of 1, and a movie with an extremely negative description would have a sentiment score of -1.

Each dataset underwent a 70-15-15 split between train, validation, and test datasets, where validation was used after a training run for a model completed, and testing ran as a separate step on a trained model. Data ordering is deterministic within, but not across models (ex. same order for all linear regression runs, but different order from any neural network runs).

Dataset	**Year**	**Genre**	**Content Rating**	**Runtime**	**Director**	**Sentiment of Desc**
Baseline	z-score	One-hot	One-hot	z-score		
Directors	z-score	One-hot	One-hot	z-score	One-hot	
Sentiment	z-score	One-hot	One-hot	z-score		Raw (by default [-1,1])

Table 1: Features included in each dataset with normalization method. If blank, not included

3 Methodology

Each model was evaluated by its Mean Squared Error and R^2 score. The R^2 score indicates the efficacy of a regression model. 1 indicates a perfect model, 0 indicates a model returning the average of all labels, and negative indicates that the model is worse than average[5][10]. Additionally, a pseudo-random model was made to give a comparative baseline against randomly picking a rating.

3.1 Linear Regression

Linear regression is less effective for complex data. However, it is simple to implement and run, and gives a good comparative baseline. Simpler models can be more resistant to over-fitting, use less computational power, and be more easily understood to external stakeholders. If a linear model performs well compared to something more complex, like a neural network, there are valid arguments for using the linear model.

Linear regression algorithms were written with NumPy to implement equations taught in the course. Several methods were used, including solving for the closed form as

$$W^* = (X^\top X)^{-1} \cdot X^\top Y \tag{1}$$

This equation was also expanded to implement Ridge Regression as

$$W^* = (X^\top X + \lambda I)^{-1} \cdot X^\top Y \text{ where } \lambda = 0.1 \tag{2}$$

Additionally, Gradient Descent, Mini-batch GD, and SGD were implemented with $\alpha = (0.1, 0.1, 0.01)$ and epoch $= (1000, 10, 10)$ respectively. All models were trained on the Baseline dataset. For the

Directors dataset, only Mini-batching was used, as its size, 1.8 GB, only allowed for chunked loading, which fortunately coincides with how Mini-batch runs. For the Sentiment dataset, only the Gradient Descent methods were used, as an inverse could not be found on the feature matrix.

3.2 scikit-learn models: Random Forest, SVR, and XGBoost

Ensemble methods, such as Random Forests, are known to be effective for regression models[8][9][2]. Initially, only a Random Forest model was going to be implemented. However, the Scikit Learn and XGBoost libraries implement other regression models utilizing nearly identical code, so it was trivial to also test SVR and XGBoost. The SVR model is a support vector machine model, and XGBoost is a Gradient Boosting algorithm[6][7][1].

While the sklearn and xgboost libraries take care of the algorithmic implementation, code needed to be written to configure data and evaluate the models, and hyperparameters had to be chosen. For the Random Forest, a depth of 15 was used after iterative testing. For SVR, a regularization value of 1 was used, a polynomial degree of 3 was set, and a kernel coefficient of 0.1 was used. For XGBoost, the default values of the algorithm were accepted as is.

3.3 Neural Network

A Feedforward Neural Network was developed. While more highlighted for their success in complex classification problems, they can also be highly effective in solving regression problems. The network was developed using the PyTorch library, using the feedforward neural network given in Homework 5 as a starting point. Data was loaded via a custom-generated DataLoader for the project data.

The neural network used 25 epochs with a learning rate of 0.01. The data was loaded with a batch size of 64. The neural network consisted of 5 layers of neuron size 62 (63 for Sentiment, 31,236 for Directors), 45, 23, 12, and 1 for the output, which represented the predicted rating[4]. All layers were linear and used ReLU for the activation function. MSE was the loss function, and the Adam optimizer was used. An Exponential Learning rate scheduler was used to modify the learning rate as training was performed. Training was performed on a Linux PC with an 8GB GPU.

4 Results

Model	**Baseline**		**Directors**		**Sentiment**	
	MSE	**R2**	**MSE**	**R2**	**MSE**	**R2**
Random Rating Generator	8.59662	-4.19021	8.59662	-4.19021	8.59662	-4.19021
Linear Regression - Closed Form	4.44555	-1.62475				
Linear Regression - Ridge Regression	4.4493	-1.62696				
Linear Regression - Gradient Descent	4.54581	-1.68394			4.43741	-1.64103
Linear Regression - Mini Batch	4.45112	-1.62804	3.73268	-1.24715	4.52935	-1.69575
Linear Regression - SGD	4.83463	-1.85447			4.70433	-1.79989
Random Forest	1.04926	0.38050			1.08369	0.35502
SVR	1.03277	0.39023			1.08148	0.35633
XGBoost	1.03904	0.38653			1.08105	0.35659
FNN	0.99080	0.39958			1.12319	0.31382

Table 2: Results from All Models across All Datasets

5 Discussion and Conclusion

All models performed better than random, but only the ensenble, SVM, and neural network models had a positive R^2 score.

In the linear models, the closed-form solution performed the best among the Baseline dataset, with Ridge Regression offering a very slight improvement. This is expected, as we can compute the optimal weights directly. The addition of the Sentiment data actually caused the model to perform worse. The Baseline Mini-batch model had $R^2 = -1.62804$ whereas the Sentiment Mini-batch model had $R^2 = -1.69575$. Adding a feature is often seen as a good thing, as it gives a model more information to work with. However, it seems that the addition of sentiment data on the movie's description did not improve predictions. The model likely made correlations with training data that did not hold for novel data. Linear Regression with the Directors data performed the best among Mini-batch as well as all linear regression models, with $R^2 = -1.24715$. The addition of directors was a marked improvement. Despite this improvement, it is still performing worse than a model that simply returns the average of all ratings.

The ensemble and SVM models showed a significant improvement, giving a positive R^2 score. For the Baseline dataset, all methods had an R^2 between 0.38 and 0.39, with SVR having a slight advantage. For the Sentiment dataset, all methods had an R^2 around 0.35, with only trivial differences. Looking at these results indicates that we're starting to reach what is likely the best we can do with the features we have. These methods are well regarded and proven to be effective even on complex data. Hyperparameter tuning may bring some improvement, but likely not enough to develop a strong model. Additionally, we see once again that the addition of Sentiment data on the movie's description only worsened the model's performance, showing that even though we want to add features, we need to be strategic about what ones we add. The Directors dataset was not able to be used, as these models are not built to handle receiving data in batches.

The Feedforward Neural Network on the Baseline dataset had the best performance among all models, with $MSE = 0.99080$ and an $R^2 = 0.39958$. We saw a drop in the Sentiment dataset's performance, reinforcing that this was not an effective feature to add. A batched training approach in the Neural Network was attempted with the Directors dataset. However, batches consistently returned validation $MSE = 36.56726$ after hours of training, so this approach was abandoned.

In conclusion, a mildly effective model was able to be generated. Despite using features that did not clearly correlate to a movie's audience rating, several models that performed better than average were able to be generated. To improve this model, a focus would need to be made on feature selection and hyperparameter tuning. Directors was a promising feature, but one-hot encoding every director in the dataset proved to be too memory intensive to be effective. Sentiment on the movie's description was an interesting path of discovery, but seemed to be too disconnected from the movie to contribute to the model's performance. Some desired features that were not easily obtained included movie medium (black and white, animation, etc.), language(s) spoken, and movie studio affiliated with the movie. Future work could involve finding methods of extracting these features from publicly available data sources. Some hyperparameter tuning was performed on the models, but utilizing frameworks such as Optuna to update hyperparameters more strategically could benefit the models as well. For example, the same structure of Neural Network was used between the Baseline and Directors dataset. Designing a network architecture that better handled the Director data could generate an even better performing model.

References

[1] *Introduction to Boosted Trees*. URL: https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[2] mccurciomccurcio. *How to choose best model for regression?* Apr. 2020. URL: https://datascience.stackexchange.com/a/73200/140277.

[3] OctopusTeam. *Full Amazon prime dataset*. Dec. 2024. URL: https://www.kaggle.com/datasets/octopusteam/full-amazon-prime-dataset.

[4] *PyTorch Linear Layer Out Features*. URL: https://g.co/gemini/share/1e602e17f15e.

[5] *r2_score*. URL: https://scikit-learn.org/dev/modules/generated/sklearn.metrics.r2_score.html.

[6] *SVC*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[7] *SVR*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVR.html.

[8] *The Ultimate Guide to Random Forest Regression*. Sept. 2020. URL: https://www.keboola.com/blog/random-forest-regression.

[9] *Types of Regression Models in Machine Learning*. URL: https://www.snowflake.com/guides/types-regression-models-ml/.

[10] Kate Wall. *Negative R2: Where did you go wrong?* Oct. 2022. URL: https://towardsdatascience.com/negative-r2-where-did-you-go-wrong-9d4f2aa84cfb.

6 Appendix

6.1 Code

All code can be accessed at GitHub at https://github.com/AGnias47/cis-5526-project. Any sources used in developing the code itself are referenced within that project.