Predicting the Score of a Movie in TMDB

Bharat Raman

Udacity: Machine Learning Engineer Nanodegree

Abstract. Found in Kaggle is a database containing approximately 5,000 different movies and their features [5]. Such features include genres, keywords, cast members, crew members, and vote average. With the thousands of unique cast, crew, production companies, and keywords, I posed the question: is it possible to predict a movie's voter average based on the different features? I attempt to answer this using machine learning algorithms provided by Amazon Sagemaker, and with the help of resources from Udacity's Machine Learning Engineer Nanodegree.

1 Introduction

1.1 Benchmark Model (Motivation)

My motivation for this project comes from another project by Ashwani Swain, who created a predictor by taking the average of the scores of 10 most similar movies to the subject movie [4]. To start, Swain chooses the following features to make comparisons with:

- genres
- cast
- crew (director only)
- keywords

For each feature, Swain makes a list of unique feature values for each feature.
For example, a genre's feature values consist of :

- action
- adventure
- fantasy
- crime
- etc

Then, looking at the genre's values in a specific movie, such as one of action and crime, Swain can make a binary array where 1's indicate the existence of a genre, and 0's indicate an absence:

$$[action, adventure, fantasy, crime, drama, romance, ...] -> [1, 0, 0, 1, 0, 0, 0, ...]$$

Doing this for each feature, Swain created a data-frame of binary arrays to set-up for comparisons.

To compare with other movies, Swain used the cosine-similarity algorithm offered by scipy [2]. The formula for comparison is as follows:

$$similarity = \frac{A * B}{||A|| * ||B||}$$

A and B are both equally sized arrays. The function provided by scipy returns a float distance between the two features. Hence, the closer the distance, the more similar the features.

With the similarities, Swain then found the 10 most similar movies to a subject movie and calculated the average of the 10 movies' ratings. He presented three tests:

Table 1. Predictions for Movies using Cosine Similarity

Movie name	Predicted Score	Actual Score
Godfather	6.95	7.10
Minions	6.50	6.40
Rocky Balboa	6.56	6.50

The results above yield an overall accuracy of 98.5%

I added an extra test, by selecting ten random movies to see if this accuracy was consistent throughout all movies. My results show the following:

Table 2. More predictions

Movie name	Predicted Score	Actual Score
Men Of War	5.87	5.40
Red Planet	6.45	5.40
Mooz-Lum	6.54	4.50
The Killer Inside Me	6.30	6.00
Winnie Mandela	6.36	5.20
Namastey London	6.60	6.60
The Chumscrubber	6.29	6.70
The Statement	6.85	5.90
Pirates of the Caribbean: On Stranger Tides	6.13	6.40
Dr. No	6.78	6.90

These results yield an average accuracy of 87.1%. Worth noting is that this score is not indicative of the algorithm's performance throughout all 5,000 movies, but it does paint a picture as to how well it can estimate a movie's score. With these results in mind, my goal is to try and achieve a similar accuracy, but through the use of supervised machine learning.

1.2 Capstone Outline

My project will proceed as follows:

1. Select and represent features as normalized decimal values
2. Create and train a model using Sagemaker and the XGBoost algorithm
3. Deploy the model
4. Evaluate the model via R^2 score
5. Tune Hyperparameters, retrain/deploy and reevaluate R^2 score

The project will take inspiration from some of the tactics Swain uses in his project, such as choosing the features: genres, cast, and crew, making a list of unique feature values, and creating binary representations of the features. On top of the selected features, I will also include production companies.

2 Feature Engineering

As stated before, the data-set consists of around 5,000 movies. Each movie contains various features such as genres, cast, crew, production companies, production countries, spoken languages, and keywords. The variable I identified as the target is the movie's vote-average from TMDb users. Looking at the distribution of ratings, as shown in figure 1, the average of the distribution was 6.17, and the variance of the distribution was 0.93

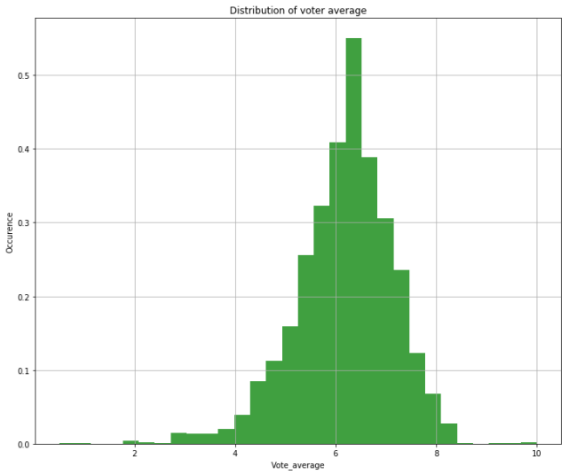

Fig. 1. Histogram of vote averages

Looking at the distribution, I believe that scores closer to the median, 6.17, will be much easier to predict due to the high concentration of movies in that area. Scores close to 0 and 10 are quite sparse and will be much harder to predict.

2.1 Feature selection

The features I've selected to use are:

- genres
- cast (top 5 billed cast members)
- crew (top 10 billed crew members)
- production companies
- keywords

I'm selecting portions of cast and crew that I think will impact the movie's score the most. Top 10 crew members will include those in direction, music composition, screenplay, and other essential roles in film production. The top 5 cast members will have the most screen-time, so naturally, they will have the most influence on a film's performance.

The data will be stored into a pandas data-frame, as shown in figure 2.

	id	original_title	genres	cast	crew	production_companies	keywords	vote_average
3682	300706	Let's Kill Ward's Wife	[Comedy]	[AmyAcker, DagmaraDominczyk, DonaldFalson, Pat...	[ScottFoley, ScottFoley]	[LostRhinoFilms]	[murderplot, wifemurder]	5.5
4424	287524	Fear Clinic	[Horror]	[CleopatraColeman, FionaDourif, KevinGage, Robi...	[AaronDrane, AaronDrane, JosephWhite, MarkBent...	[AnchorBayEntertainment, DryCountryFilms, Movie...	[doctor, fear, phobia]	4.0
3629	13996	Bottle Shock	[Comedy, Drama]	[AlanRickman, BillPullman, ChrisPine, FreddyRo...	[JodySavin, RandallMiller, RandallMiller]	[IntellectualPropertiesWorldwide, ZinniaFilmP...	[wine, winegarden, winzer]	6.6
3919	15582	Teen Wolf Too	[Comedy, Family, Fantasy]	[EstelleChandler, JasonBateman, JohnAstin, KimDa...	[ChristopherLeitch, CynthiaMcCormac, HarveyRos...	[AtlanticEntertainmentGroup]	[teenager, werewolf]	3.8
1592	4105	Black Rain	[Action, Crime, Thriller]	[AndyGarcia, JohnSpencer, KateCapshaw, Michael...	[DianneCrittenden, ElenMirojnick, HansZimmer, ...	[Jaffe-Lansing, ParamountPictures, PegasusFilm...	[japan, japanesemafia, yakuza]	6.2
1574	6780	Baby Mama	[Comedy]	[AmyPoehler, DaxShepard, GregKinnear, Sigourne...	[AvyKaufman, BruceGreen, DarynOkada, JeffRichm...	[Michaels-Goldwyn, RelativityMedia]	[pregnancyandbirth, surrogatemother]	5.8
2599	15237	De-Lovely	[Drama, Music]	[AshleyJudd, JonathanPryce, KevinKline, KevinM...	[AndrewGrant, CharlesWinkler, EveStewart, Irwi...	[Metro-Goldwyn-Mayer(MGM), PotboilerProduction...	[biography, hollywood, musical, theater]	6.3
3966	26039	Point Blank	[Action, Crime, Drama, Thriller]	["CarrollOConnor", AngieDickinson, KeenanWynn,...	[AlbertBrenner, EdwardWoehler, GeorgeW.Davis, ...	[Metro-Goldwyn-Mayer(MGM), WinklerFilms]	[alcatraz, organizedcrime, revenge]	7.1
3890	10162	Waking Ned	[Comedy, Romance]	[BrendanDempsey, DavidKelly, FionnulaFanagan,...	[AlanStrachan, AlexandreHeylen, GlynisMurray, ...	[Canal+, FoxSearchlightPictures, TomboyFilms]	[beguilement, falseidentity, independentfilm, ...	7.4
690	497	The Green Mile	[Crime, Drama, Fantasy]	[BonnieHunt, DavidMorse, JamesCromwell, Michae...	[DavidValdes, FrankDarabont, FrankDarabont, Fr...	[CastleRockEntertainment, DarkwoodsProductions...	[1930s, basedonnovel, blackpeople, deathrow, d...	8.2

Fig. 2. Sample of Data I'm Using

To process the features, the first step is to create sorted lists for each feature's values.

2.2 Creating feature lists

For this step, I referred to Swain's strategy of creating lists of unique feature values. However, I differed here in that I sorted the values by the average rating of movies they are associated with. For example, Leonardo Di Caprio would place further in the list than Nicholas Cage because since Caprio had been casted in more well-received movies. Distributions can be seen in figure 3.

Analyzing the data, I decided to remove genres from the list of features to use; because of its very low variance, the data will give me negligible information as to what rating a movie could be. Therefore, the resultant features I chose to use are: cast, crew, production companies, and keywords.

2.3 Creating Binary Arrays

Now once I've set up the feature lists, I again refer to Swain's strategy and create binary array representations for each feature. The beauty of this process is that, since the lists are organized from lowest rated to highest rated, the binary

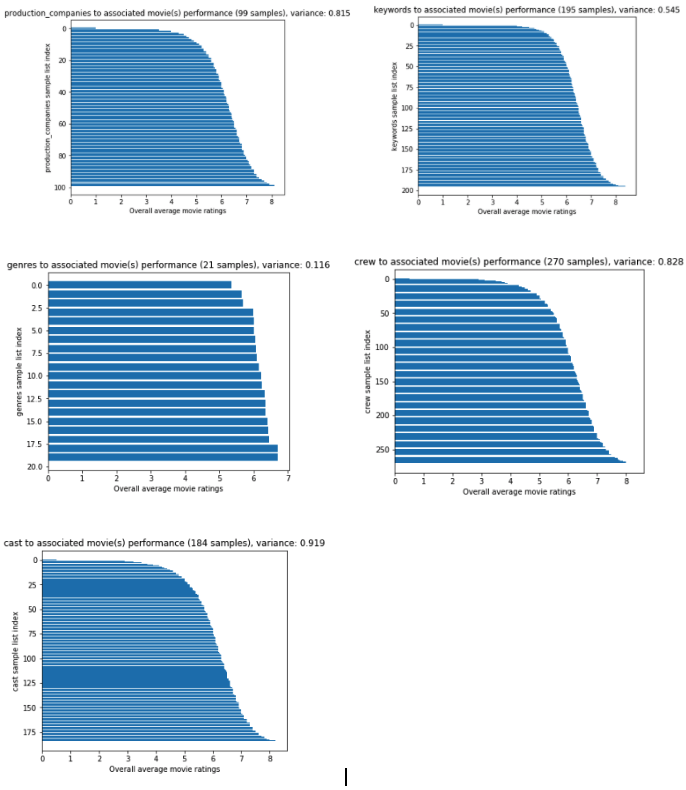

Fig. 3. Distribution of features from lowest rated to highest rated

array will show a visual representation of the quality of features associated with a movie. Figure 4 shows that binary distributions for features have a visual correspondence to to the rating of the movie; 'Don McKay', rated at 5.4, has features leaning towards the left (lower-rated features). 'Flipped', rated 7.4, has features leaning towards the right (higher-rated features).

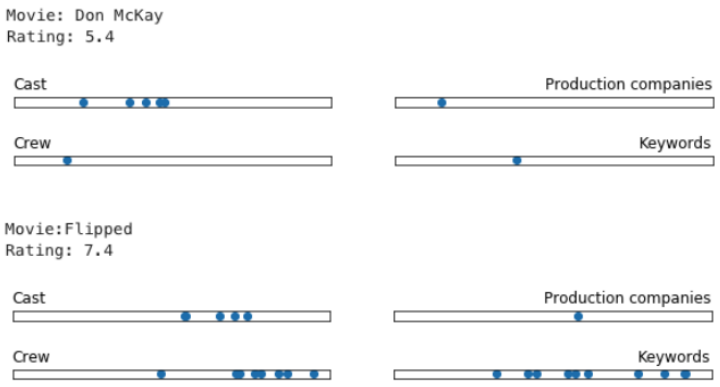

Fig. 4. Visualization of binary arrays in a number line for two different movies

2.4 Converting binary array into decimal representation

Finding concentration points among multiple regions: In order to convert the binary array into a decimal representation, I wanted the decimal value to be a point in the number line that represents the distribution of 1's in the binary array. To do this, I separated each array into groups of 50, and looked for concentrations in each group. Most of the groups had no 1's, and were ignored. For the groups that did, I saved the average index and number of 1's as a tuple, to be used in the next step.

Creating the decimal representation: To create the decimal representation, I simply took the list of tuples for each array, and calculated the weighted average:

$$point = \frac{\Sigma pos_i * n_i}{\Sigma n_i}$$

- pos is the position of concentration within the i^{th} group
- n is the number of 1's found around pos within the i^{th} group

This provided a point that best represented the distribution of 1's in a binary array. Figure 5 shows the features represented as decimal values.

	id	original_title	cast	crew	production_companies	keywords	vote_average
2040	2176	The Glass House	4132.8	4252.700000	1434.000000	3831.428571	5.4
1650	10350	Wing Commander	2540.2	2951.400000	390.000000	2777.000000	4.0
689	14306	Marley & Me	4281.0	7333.777778	1902.500000	4401.571429	6.9
3159	30497	The Texas Chain Saw Massacre	7514.2	9980.166667	3275.500000	5411.187500	7.2
678	300168	天降雄師	4191.6	4564.555556	1992.777778	1295.000000	5.9
1502	11978	Men of Honor	5111.2	7216.363636	2362.000000	3440.500000	7.0
2756	10691	Crazy/Beautiful	4601.2	6490.100000	2871.000000	5233.545455	6.6
380	921	Cinderella Man	5500.6	9162.000000	2918.000000	6637.333333	7.3
1334	23742	Cop Out	2506.2	4999.250000	2634.000000	3341.200000	5.3
4592	3059	Intolerance	8579.8	12689.888889	4647.500000	7216.076923	7.4

Fig. 5. Decimal representations of features in data-frame

Normalization and visualization: Once I created decimal representations for each feature, I then normalized the values to fit between 0 and 1. When I visualized the distributions of the normalized features, I was pleased to see a very visible correlation between the features and ratings, as seen in figure 6. One thing to note is the lines that stand out in the 'production companies' and 'keywords' plots. These points represent absent features, in which a movie does not have a documented production company and/or keyword. I tried to remove these points, but a considerable amount of movies (approximately 500) were in that category. Therefore, I only removed points that had no data whatsoever (no cast, crew, production company, and keyword), which only removed 8 movies from the data set.

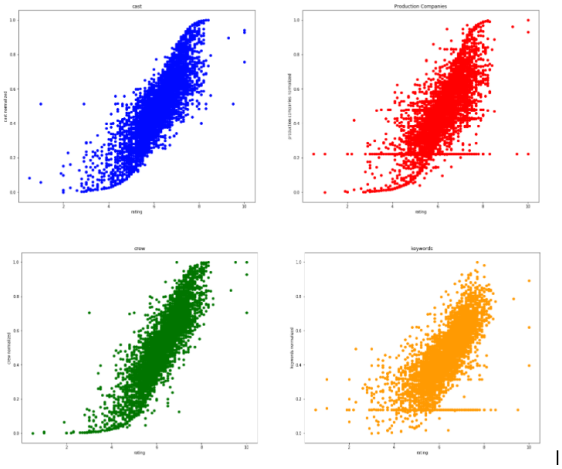

Fig. 6. Visualization of normalized features to movie rating

This concludes the feature-engineering segment of the project. The next step is training an XGBoost model.

3 Training and Deploying with XGBoost

3.1 Splitting Data

Before introducing XGBoost, I had to first prepare the data. To do so, I split the data into three partitions: Training, testing, and validation. The distribution can be seen in table 3.

Partition	Percentage
Training	70%
Testing	15%
Validation	15%

Table 3. Splitting Data into Training, Testing, and Validation

3.2 The XGBoost Model

For this project, I will be using XGBoost to train and deploy my model. XGBoost is ideal for regression-based learning due to its flexibility in handling various datatypes, combining multiple weaker models, and using various hyperparameters [1]. Its flexibility in hyperparameters and their ranges makes XGBoost easy to improve on via hyperparameter tuning, which I implement later in this project.

How XGBoost Works: XGBoost uses gradient boosting, in which a collection of weaker models (regression trees) are combined to increase accuracy of the fit. During the training process, more tree models are added to predict errors of previous trees and minimize loss. Therefore, loss is gradually reduced as training proceeds, resulting in a more accurate fit.

There are also many references notebooks available on different stages of machine learning with XGBoost. One such reference I used was a tutorial from Udacity, which trained, deployed, and tuned an XGBoost model to predict prices in the Boston Housing Market [6].

3.3 Training

For setting up my XGBoost for training, having referred to the parameters used in the Boston housing tutorial, I used the following Hyperparameters:

- max depth = 5,

- eta = 0.2
- gamma = 4
- minimum child weight = 6
- subsample = 0.8
- objective = linear regressor
- early sropping rounds = 10
- number of rounds = 200

Next, is the actual training. Worth noting is that the linear regressor was replaced with a squared-error regressor. The train method completed with a final training root-mean=squared-error (rmse) of 0.35157 and a validation rmse of 0.423419.

3.4 Deployment

I then created an endpoint by calling the deploy method. The deployment will return a predictor, which will be used to visualize predictions and create an R^2 score.

4 Predictions and R^2 score

Once the endpoint is created, I then accessed it to create an array of predictions from the test features, which can be compared with the test target (movie ratings). Shown in the Boston Housing Tutorial, the predictions can be visualized by creating a scatter-plot with the axes target test and test predictions [6]. For the best predictions, the scatter-plot should fit around the line:

$$y = x$$

The resultant plot, figure 7, shows that the points do indeed fit close to that line.

For a quantitative measurement, I calculated the fit's coefficient of determination, otherwise known as the R^2 score. The score is best defined as a measurement for how well the variance of the dependent variable (ratings) can be explained by the independent variables (features: cast, crew, production companies, keywords)[3]. In this case, the R^2 score was calculated to be about 0.806

0.806 is a fairly good score for this fit. This shows that machine learning is in fact a viable way to make predictions on a movie's score; however, it is possible to get an even better score if the model is tuned via hyperparameter tuning

5 Hyperparameter Tuning

For this step, I utilize hyperparameter tuner offered by Sagemaker to tune the XGBoost model. My objective is a minimal root mean square error for the validation dataset. The tuner will be doing 20 training jobs, with three at a time done in parallel. The hyperparameters being tuned are:

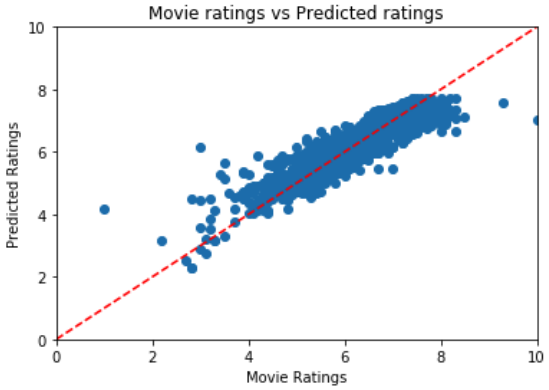

Fig. 7. Visualization of test predictions to test targets

Table 4. Hyperparameters being tuned

Hyperparameter being tuned	Parameter Type	Range
Max Depth	Integer	3 to 12
eta	Continuous	0.05 to 0.6
minimum child weight	integer	1 to 9
subsample	Continuous	0.5 to 0.9
gamma	Continuous	0 to 10

After all 20 jobs are completed, I deployed the best model, which had a validation RMSE of 0.400583. After making the new predictions, the resultant visualization, as shown in figure 8, indicates that the points follow the line $y = x$ more closely.

The resultant R^2 score yields 0.829, which is a great improvement from the previous score of 0.806.

6 Conclusion

In retrospect of about 5,000 movies, an R^2 of 0.829 shows that a movie's rating can indeed be predicted, to a reasonable accuracy, via supervised machine learning. The results are comparable to those in Swain's Cosine Similarity model; however, I've managed to evaluate an accuracy for movies within the testing data, while the accuracy I calculated from Swain's model comprised of just 10 randomly picked movies.

6.1 Comparing with the Benchmark

Swain's model, without incorporating the use of machine learning, was accurate to about 87% among ten randomly selected movies. However, the score is not representative of the entire data-set; some predictions falter below 60% accuracy.

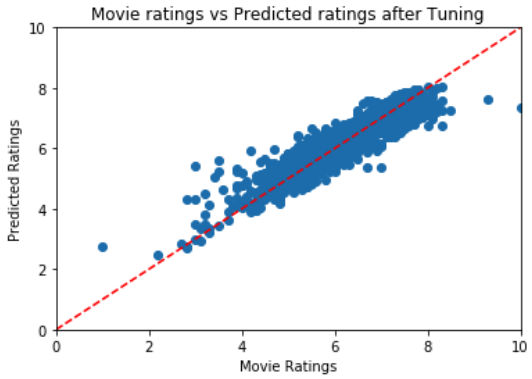

Fig. 8. Visualization of test predictions to test targets after hyperparameter tuning

Moreover, it is inefficient to predict all movie ratings in the data set, as each prediction takes about 13 seconds (using cosine similarity among four features, three of which are arrays of length 1000+, and with all movies in the data-set to find the top 10 similar movies). Therefore, Swain's model is not ideal for finding an overall trend between ratings and features (It's doable, if you can set aside about 17 hours to run every prediction).

Through machine learning, I believe I've improved upon Swain's model by identifying a trend between features and movie ratings; as shown in my predictions and R^2 scores, my model shows a functional relation between a movie's ratings, and its features, where approximately 83% of the variance in ratings can be explained by the feature values.

6.2 Final Thoughts

I think there's room for improvement for how I prepared my features; the decimal value was calculated as the weighted average of concentrations in a binary array. The limitation of this is that the weighted average is a single point describing the distribution of 1's in an array of length 1000+, and thus loses information from the features. Perhaps there is a more accurate way that retains the information from the features.

As for my hyperparameters in the XGBoost model, I believe that by experimenting with different variables, and their set values, I could boost the starting R^2 score before hyperparameter tuning. Same goes for the hyperparameters used for tuning the model, in order to get a better final R^2 score.

Another thing worth noting is that I did not attempt to use Sagemaker's Linear Learner model, which could have possibly yielded a better score.

I implore others to work off and improve on this project and possibly apply this to practical applications.

7 Acknowledgements

I'd like to thank:

1. Ashwini Swain for providing a neatly presented benchmark model for me to work off on
2. my Udacity mentor Justin K for providing feedback throughout the capstone.
3. Udacity for a great introduction to the field in Machine Learning in the Machine Learning Engineer Nanodegree.

References

1. AWS: Xgboost algorithm, https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html
2. Scipy: scipy spatial distance cosine, https://www.kaggle.com/kernels/scriptcontent/9804615/notebook
3. sklearn: Model evaluation, https://scikit-learn.org/stable/modules/model$_e$*valuation.htmlr*2−*score*
4. Swain, A.: What's my score?? (2019), https://www.kaggle.com/kernels/scriptcontent/9804615/notebook
5. TMDb: Tmdb 5000 movie dataset (2017), https://www.kaggle.com/tmdb/tmdb-movie-metadata
6. Udacity: Sagemaker deployement on github, https://github.com/udacity/sagemaker-deployment/tree/master/Tutorials