Predicting Movie Performance

Using Machine Learning to Predict Movie Revenue

Kayvan Jalali[1], **Carlos Puerta**[2]

[1,2] Data Science and Analytics Program at the University of Maryland, College Park, MD 20742

December 13, 2022

Contents

1. Introduction

Predicting movies can be an exciting task; it allows us to delve into the complex world of film and explore the factors that drive a movie's success. Whether you are a movie buff, a data scientist, or just someone who enjoys trying to figure out why some movies do well at the box office while others flop, predicting movies can be a fascinating and rewarding pursuit.

If we are able to better understand the complex factors that drive a movie's success by analyzing data on factors such as a movie's budget, genre, and cast, we can gain insight into the characteristics of a successful movie and learn what makes some movies more likely to do well at the box office than others.

2. Methodology

In order to create the machine learning model, we are going to grab information from The Movie Database (TMDB). TMDB offers a public API that we can use to request a large amount of data on past, and upcoming movies. This database is constantly updated and is adding new movies and TV shows. Our goal is to have our model grab the most up-to-date information on requested movies and use that information to predict the performance. We are going to make a variety of machine learning algorithms to try and find the most accurate model to predict the revenue of movies, both past and future.

Looking at the information that can be requested from TMDB, we have some attributes that we believe might be helpful in predicting movie revenue. Specifically, we will be looking at time since release, genre, average score, popularity, number of votes, and crew/cast members. We believe looking at these attributes we will be able to come up with a good approximation to the actual revenue of the movie.

It will be important to look at the distribution of the data. As one might imagine, there have been movies that have been unbelievably popular and their data points will likely be a very extreme outlier. Having this in mind, it will be imperative to pay particular attention to these cases, as they will make our model worse if unaccounted for. Furthermore, there are very big discrepancies in the size of cast and crew between certain movies. Smaller films have a small team, but big films have a disproportionately large team, for this reason, we consider only modeling data for the top cast and crew per film.

In terms of modeling, we will attempt several analyses. First and foremost, will graph our predictors versus revenue to get an idea of any correlations that are visually apparent. From here we intend to use multiple regression, multi-layered perceptron regressors, and gradient boosting regressors to see what the best model is. If the resulting model is not good, we will consider modeling a logistic regression to try and classify movies into either successful or unsuccessful, and then having separate models within these two categories to model revenue.

Once our model is set up, we will be displaying all information on a website. This website will be made using Django for the back-end, which will house all of the modeling and machine learning, and HTML, CSS and JavaScript for the front-end. The website will display an assortment of random movies that we will predict with our model. In addition, the website will display a couple recent movies, and a couple of upcoming movies–again these will include the predicted revenue. Finally, the website will also have a search bar, so that users may search for any particular movie and see how the algorithm performs.

3. Data Source

As previously discussed, we will gather the data from TMDB. TMDB allows you to query their database using keywords, or you may request a specific movie using an identifier. As of writing this, TMDB does not allow you to request multiple specific movies using their identifiers, as such we will be running a script to request movies individually, and storing them locally so that we can access them at any given time. Since the TMDB API is a REST API, we opted to keep all of the data returned in case we decided to use anything else. This data is stored directly in JavaScript Object Notation (JSON) format and split up into smaller JSON files for performance.

In Figures 1-4 we plotted some of the variables against each other and shaded them all in terms of centered revenue to identify any noticeable characteristics. Looking at the plotted data we see the movies with higher popularity, total popularity, vote average, vote count, budget, and number of crew tend to have higher revenue. There is not a steady increase but after a certain point the movies generate much more revenue than the others.

Working with the data, we have two approaches, one for the machine learning algorithms and one for the website. Both methods read the data as JSON and clean it. For machine learning we convert it into a Pandas data frame, and for the website we convert it into custom python classes. Pandas data frames allow us to leverage libraries like Sci-kit Learn in order to quickly and efficiently train different models. The python objects make it easy to share data between different Django views, which ultimately make it much easier to move information from the back-end to the website.

From the TMDB database we pulled the following information: backdrop path,

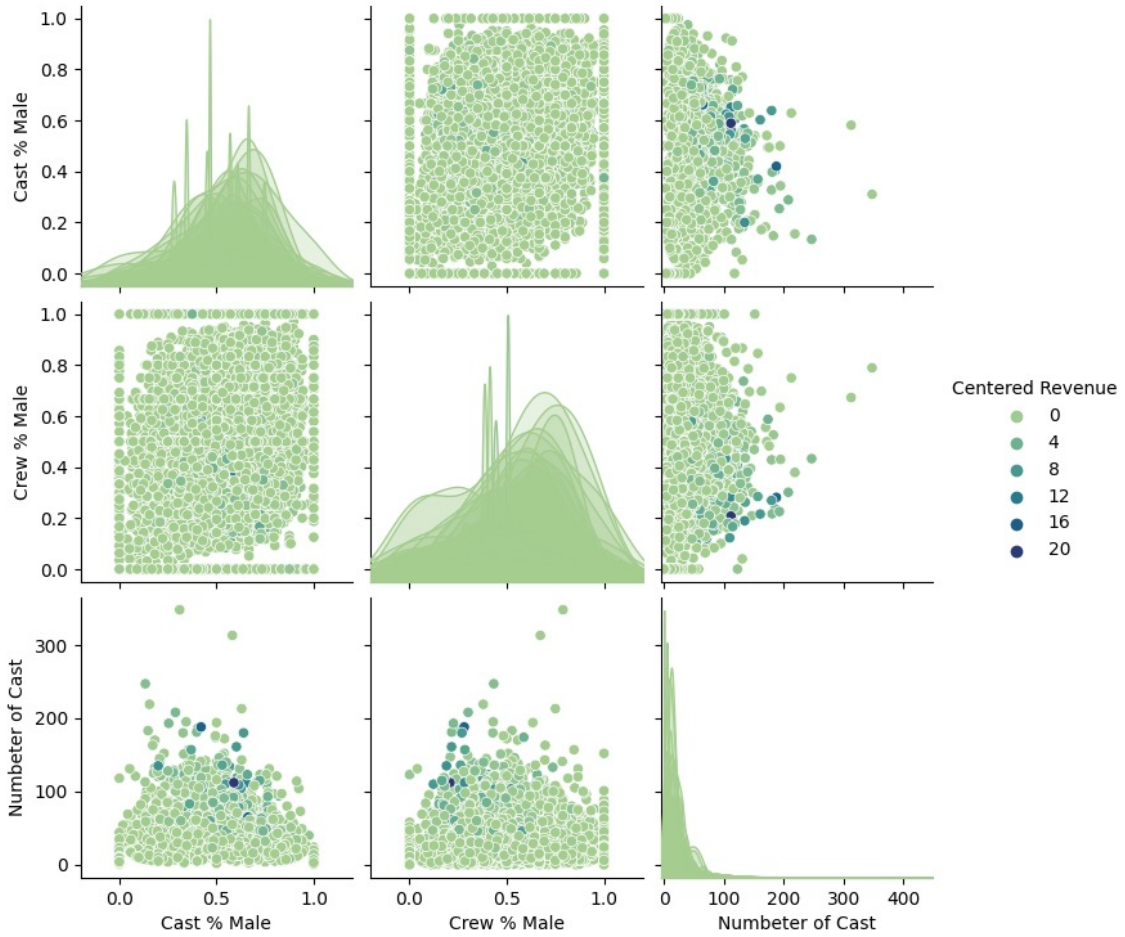

Figure 1. Variable breakdown and relationship examination with centered revenue

collection, budget, genres, homepage, movie id, original language, original title, overview, popularity, poster path, production companies, production countries, release date, revenue, run-time , spoken languages, status, tagline, title, video, vote average, vote count, and credits into a JSON file then converted into a pandas data frame. We dropped missing values and filtered movies that are released and have revenue greater than zero.

Some variables were more complex and needed extra cleaning. Release date was changed into a date-time object and also used to create a new variable for the amount of days a movie has been out. Revenue and budget were standardized by subtracting the mean and dividing by the standard deviation to make interpretation easier. Furthermore, we created new variables that stored the number of different

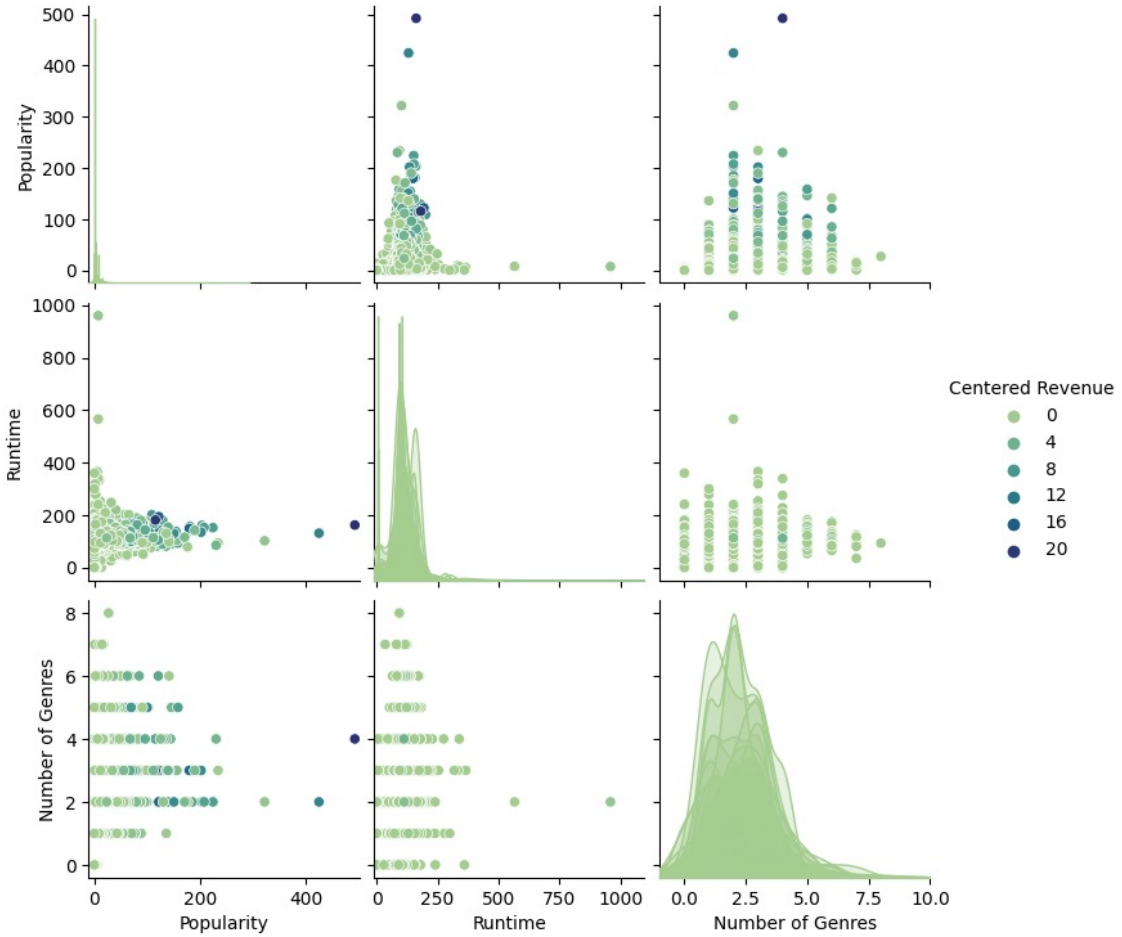

Figure 2. Attribute relationships and centered revenue

genres, the percentage of male cast and male crew, the number of cast and crew, and total popularity of the cast and crew. Finally, we created a variable to mark movies as extreme outliers, this variable "anomaly" was true for movies whose revenue was 1.5 times the interquartile range.

Table 1

	Budget	Revenue	Popularity	Vote Average	Voter Count	Days Out
Average	16,611,050	50,304,330	13.63	6.15	1,110	8,442
Standard Deviation	33,246,980	134,646,300	17.58	1.34	2,528	6,362

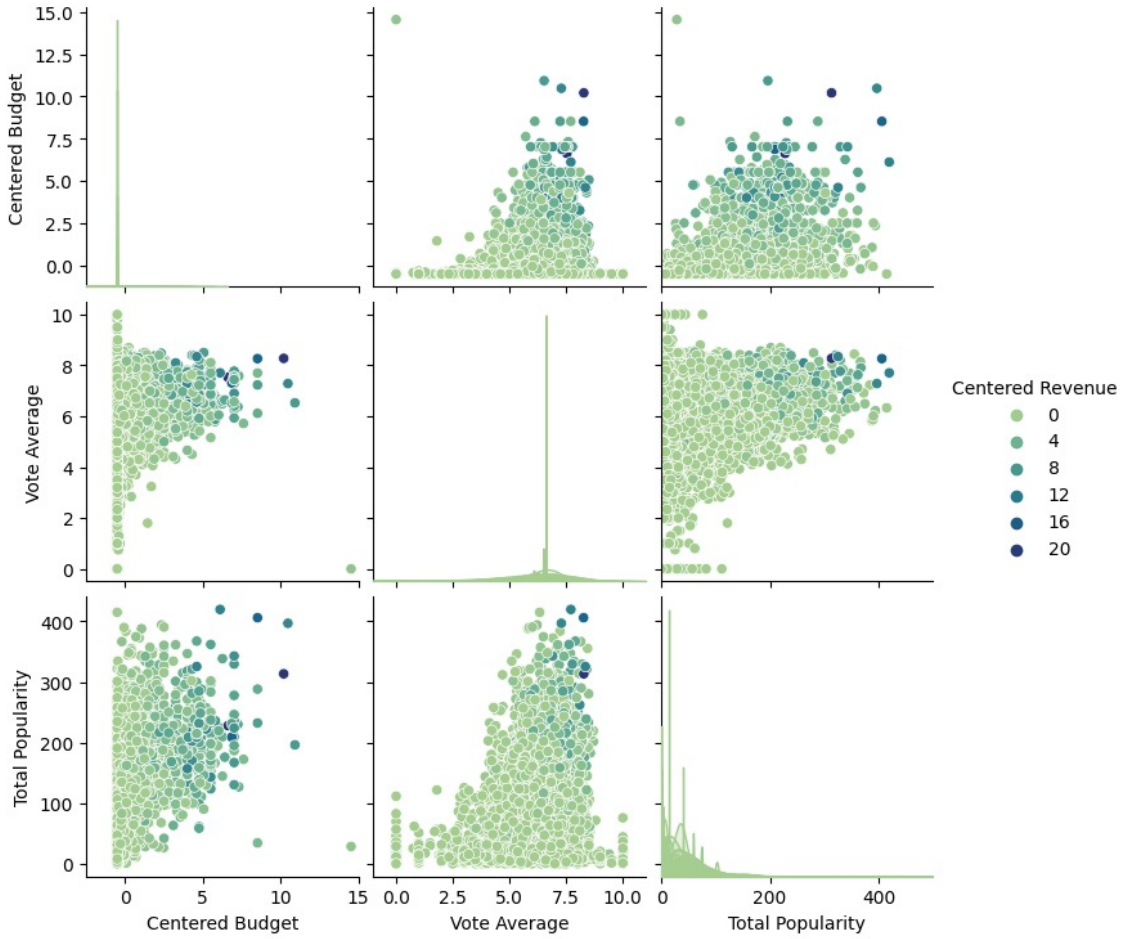

Figure 3. Trend in budget increase in revenue and vote average

4. Algorithms and Code

The first model we created was a multiple regression model to predict centered revenue with the features centered budget, vote count, popularity, and anomaly. This model was quickly implemented to see if there could be a relatively simple association between the features and target.

We also modeled using a Multi-layer perceptron regressor (MLP). MLP optimizes the squared error using the limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) or gradient descent. The model trains iteratively as the partial derivatives of the loss function with respect to the model parameters are used to update the parameters. We set the solver as LBFGS, alpha as 1e-5, and hidden layer

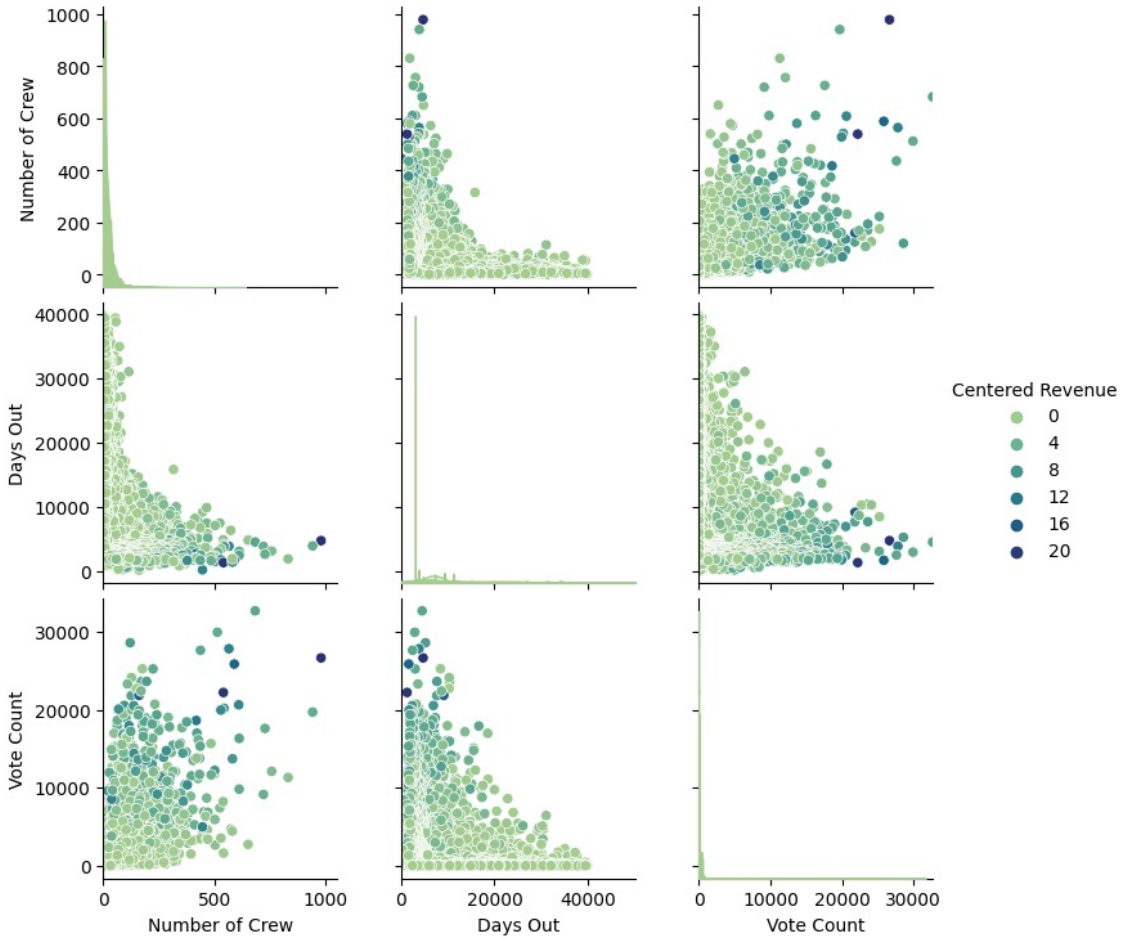

Figure 4. Increase in vote count correlated with increase in revenue

sizes as an array of fifteen. However, with this model it was clear that some of our features were not contributing to the overall model, and so we decided to discover the most important features.

To do so we implemented a gradient boosting for regressor. Gradient boosting starts with the mean value for the target variable, then calculates the residuals by measuring the distance between the mean and the value of the target variable. A decision tree is then created using the features to predict the residuals. The mean is added to the output of the tree to create new predictions. New residuals are created with the predicted values, a tree again is created from those residuals, and predicted values are generated. This process is repeated until defined parameters are satisfied.

With gradient boost we predicted centered revenue with the features centered budget, days out, number cast, number crew, vote average, vote count, popularity, total popularity, anomaly. We set the number of estimators as 100, learning rate as 0.1, max depth at 1, and loss function as the squared error. After building the model we looked at the feature importance and saw centered budget, vote count, popularity, and anomaly were the most important features based on mean decrease in impurity.

With this new information, we created 6 additional models to try and find the best one based completely off of the new variables that were deemed relevant and important.

5. Evaluation of Models

First, we tested three different models against each other–again we used multiple linear regression, MLP regressor and gradient boosting regressor. All of the models used the same training data, with the same features identified previously.

The multiple regression model was the simplest of the three, and performed very well with a score of 0.766 and a mean squared error of 0.194. The MLP regressor model had a slightly worse score of 0.719 and a mean square error of 0.233. The gradient boosted regressor model had a score of 0.790 and a mean squared error of around 0.174.

Next we tried to see if making our feature space orthogonal would have an effect on our model accuracy. We used Primary Component Decomposition (PCA) to reduce our features to just two, and used these two new features to train the machine learning models again.

Table 2. Machine Learning Models' Score and Mean Squared Error (MSE)

	No PCA (4 Features)		**PCA (2 features)**	
	Score	MSE	Score	MSE
Multiple Linear Regression	0.766	0.194	0.761	0.198
Gradient Boosting Regressor	***0.79***	***0.174***	0.81	0.158
MLP Regressor	0.719	0.233	***0.817***	***0.152***

Using PCA, our multiple regression remained nearly identical with a score of 0.761 and a mean squared error of 0.198, our MLP regressor improved to 0.817 with a mean squared error of 0.152, and our gradient boosted regressor also improved to 0.81 with a mean squared error of 0.158.

The best model we came up with used the MLP regressor using two primary components. However, we decided to use the Gradient Boost model with the original features, as it performed faster with our back-end, with very little loss in in terms of accuracy when compared to the best MLP regressor model.

As such, the final model used on the website uses a gradient boost regressor and has a score of 0.79 with a mean squared error of 0.174

6. Website and Hosting

For the website, we used Django for the back-end. Django allowed us to leverage popular python machine learning libraries, while also allowing us to put our efforts into building the best predictive model we could instead of focusing on the building of a website. Alongside Django, we also used a website template from the popular page HTML5 UP. The HTML5 Up template was critical, as it offered us a professional looking template that is fully responsive, meaning it is fully accessible on mobile, and cut down drastically the development time that would have been needed otherwise. With these two tools we put together a professional looking website that satisfied all of the requirements of the project.

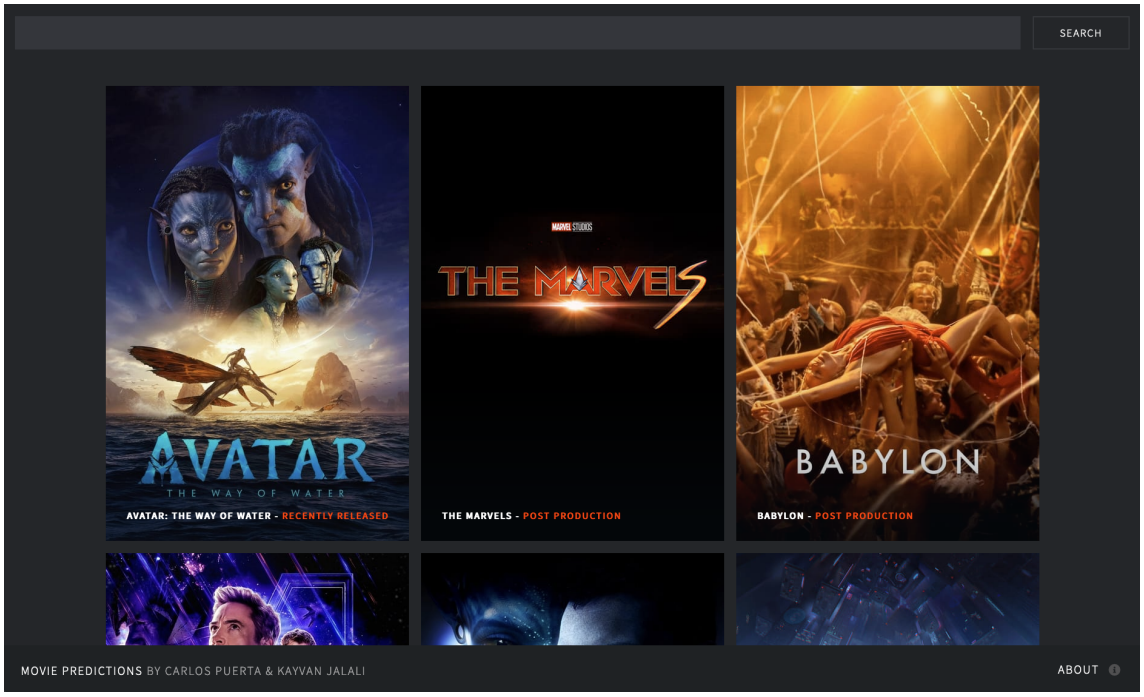

Figure 5. Homepage of Movie Predictor Website

The website compliments the machine learning models by presenting information that is clear and to the point with three main sections. To showcase the performance of the predictive algorithm, a section of the website was dedicated to displaying random movies alongside their performance versus what the machine learning model predicted. This portion of the website is very important, as most users will likely search for better known movies that have been wildly successful. As such, showcasing lesser known films allows the algorithm to shine under an unusual spotlight.

Section one of the website showcases movies that are already released, so an entire section has been dedicated to some of the most popular upcoming movies. While the performance of these movies cannot be checked, it is important to see how the algorithm is predicting movies that are not yet released.

The final section of the website allows users to search for specific movies through keywords. Because it is very likely that users will want to check the performance of the machine learning model with regards to a specific movie, it was very important

that the users are given a chance to do exactly that. As a result, this section is very similar to the first section with the key difference that the movies are hand picked by the user.

Figure 6. Movie Description Page

Because of the complexity of the website, and the setup required to run the server locally, it was very important to host this project on a cloud server. To this end, we decided to use railway.app, which is an up and coming host with a reasonable free tier. The website is currently hosted at this address (https://web-production-f732.up.railway.app), and can be accessed anywhere. It is important to note that the free tier of railway.app only includes access to 500 hours of server use per month, which is about 220 hours short of the usual month. For this reason, it is possible that if one were to attempt to access the website after the 20th day of the month, they might find that the website is down. In the event that this happens, the user can wait until the 1st of the next month, or they can contact us.

The source code for the website can be found on Github at https://github.com/KayvanJ/final_project_602. Because of hardware limitations with the rail-

way.app free plan, the hosted website exclusively has the data used for training and the single final model used for predictions.

7. Constraints and shortcomings

While the finalized model is really good at predicting certain kinds of movies, its performance varies a lot. Perhaps the biggest factor contributing to this is our data source. We started with 277,000 movies in our data set, but a simple filter, total revenue greater than zero, reduced this number to just under 12,000 movies, or slightly less than 5% of our original data. With such little data, we were unable to train on a sizable number of data, and we did not feel comfortable trying to fill in the gaps. The incompleteness of data was likely due to the fact that TMDB is a community based database, but also by the fact that many movies (particularly smaller ones) do not publish information on their performance.

Another shortcoming of the model was the value of time into predictions. As one might imagine, movies tend to have a stronger performance around the time they release, particularly if they are released in theaters. Because of this phenomenon, movies released more recently had a disproportionately large effect on the prediction of overall revenue. Movies that had been released for only a few weeks or months, were being predicted as outperforming movies that had been released for much longer–and while this is many times the case, it was very obviously overcompensating and incorrect the majority of times. This is the reason why we ultimately decided to drop time as a predictor.

Finally, we were able to visualize and observe the fact that certain movie genres performed better than others. We wanted to somehow build genre as a factor into our predictive model, but were unable to come up with a logical numerical format in which genres should be stored and rated. We are confident that building this

into a future implementation of the model would help improve the accuracy of the predictions.

8. Conclusion

In the end, we found that predicting movie revenue was a lot more difficult that we had envisioned. We were greatly limited by a lack of experience and understanding of the field. Nevertheless, we were able to come up with a decent machine learning model that we are proud of, and we learned a lot making everything. In the future, we look forward to reading up on other approaches used for this particular task (such as a Random Forest decision tree) in order to improve and refine our approach to this task.