Let's All Go To The Movies

Joseph Wilkes, Jacob Rohde, Pauline Caussé,
CS472, Fall 2019
Brigham Young University

***Abstract*—This paper analyzes how different machine learning models can predict a movie's domestic and worldwide gross revenue based on common features. To answer this question, we compared the performance of three different models and looked at the resulting r-squared score. Our results showed that K-Nearest Neighbor and Backpropagation were the models that fit our data the best. However with more computational power and time, the Random Forest could have had comparable results.**

I. Introduction

Predicting how much revenue a movie will produce has been an important question especially for investors. In this project we will attempt to create an accurate model that can predict the revenue of a certain film based on simple, public features. We hope that by creating this model we can help investors make safe investments or at least understand how strong their investment is in comparison to other movies in the past. In this paper we will discuss the progress that we have made in creating the best model to predict movie revenue. Including data sources, data collection, and limitations that we have found in researching this problem.

II. Methods and Data

We had to get the data from several sources. The web sources were primarily The Numbers, International Movie Database (IMDB) and the Movie DB. We'll describe these sources more thoroughly.

After collecting the data we explored it to build an understanding that could potentially improve predictive effectiveness.

A. Data Sources

When we started this project we knew about 'the numbers.com', which had a list of the top grossing movies with revenue data. We were able to collect this data by copying and pasting it 500 at a time. Fortunately, this maintained the correct formatting when storing it. After we had our output class, we needed a way to correctly match each data point from the-numbers to an IMDB instance in order to accumulate additional features. Collecting these features from IMDB manually would have been very tedious and would have taken too long. We began to look into the different ways to gather information from IMDB.

Initially we considered downloading the data from the IMDB website, which they have available in large comma separated variable files. Eventually we decided that this approach was complicated and that we should see if we can find a more simple solution.

We then began to look and see if the IMDB had an Application Program Interface (API) that we could use to gather the correct attributes. IMDB does not have an exposed API, however a site called omdbapi.com has an exposed API that we can use to collect information from IMDB. The only draw back to this API is that the site limits you to 1,000 API calls a day. This turned out to be a real problem, as the-numbers and IMDB fairly seldom agreed on a title, leading to both a search to find the correct id, and also a challenge in getting it. The omdbapi API also would only return one result when searching for a movie instead of a list. So we would most likely be searching for a movie more than once. This process would at least double our limited number of calls to this API. So we decided that the best way to do this was to use theMovieDatabase (TMDB) which has an exposed API that also lets us call it as many times as we want. We created a script that would run through all the-numbers data points and search TMDB for the title, if only one was found we would automatically record the IMDB ID for that movie. If no title or more than one was found, the script would prompt us to take action and be the deciding vote for which IMDB ID we would store in a file. We could not use fuzzy matching to find a best match because the wrong movie could match better than the correct movie and that some movie titles in the-numbers were in a different language than English. This worked pretty well, we could still only search for 1,000 movies a day, but this script would resolve naming conflicts and remove a lot of the man power that was needed in collecting our data. After three days of gathering data, we decided that 3,000 data points was enough to start training our models.

The main issue with our data set is that when we collected data from Numbers, we had no way to randomize our selection of movies. Meaning that we only selected the top 3,000 grossing movies instead of a even distribution of all movies, whether they were successful or not. This can lead to our results being skewed and give movies a higher revenue estimation than they deserve.

B. Exploratory Data Analysis

Our data set included many fields that we needed to alter or drop before we could begin training our models. Immediately, we could get rid of the title of the movie and the IMDB ID, as both of these fields were for indexes and did not have any information about the revenue of the movie.

In order to build our models in the most efficient and effective way, we analyzed the features and their relationships.

Primarily, we needed an understanding of the potential targets. We had two options for a target value and different models tested each target. Option one was the domestic revenue, and option two was the worldwide revenue. The values of these targets range from hundreds of millions of dollars to billions of dollars.

These targets are visualized in Figures 1 and 2. A log scale is also useful for visualizing the biggest values in the target in relation to the entire distribution.

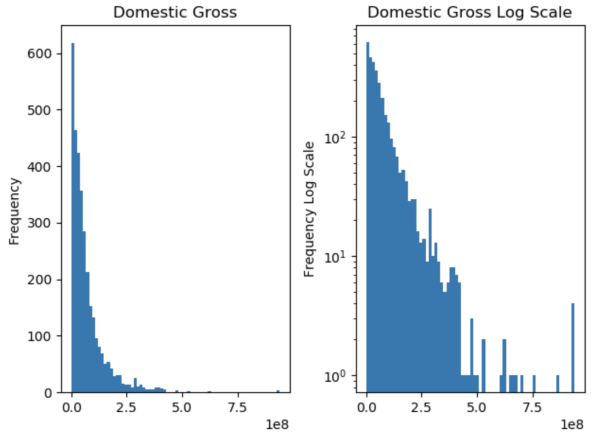

Fig. 1. Domestic Gross Revenue Visuals (Log Scale Right)

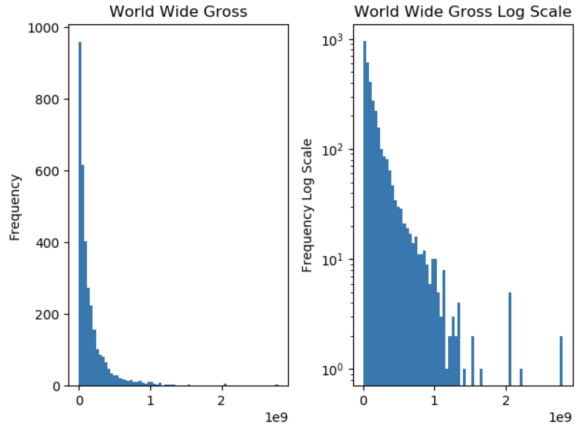

Fig. 2. Worldwide Revenue Visuals (Log Scale Right)

Another critical aspect of understanding the data was to identify which features had missing values. A horizontal bar chart (Fig. 3) displays the count of missing values by feature. The target null value counts are shaded in blue.

It was useful to visualize the distribution of several features like Genre, and what the movie was Rated as a preliminary step to identifying the importance of each feature.

For Genre, the problem was that a single movie could have between zero and five genres. Because of this, we decided to take a couple different approaches and create multiple different data sets. One solution was to create a top three genres data set. This approach would create three columns and create a mapping between ids and genres. The problem with this

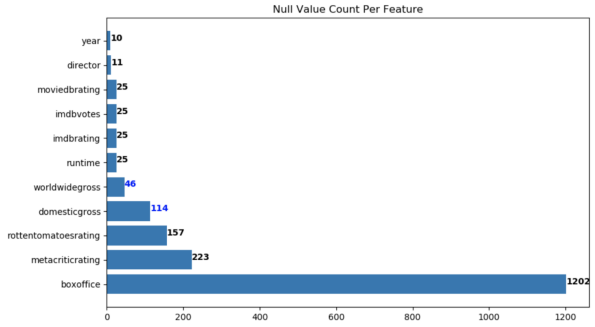

Fig. 3. Null Values by Feature

approach was that we did not know which genres were more important to the movie in the case of a movie having more than five. We could not find any documentation that said the IMDB would list them in any particular order. At least one model encoded Genre into a columnar binary feature, where each column represented one genre. Most movies had several genres. For example the movie Avatar (2009) is classified as Action, Adventure, Fantasy, and Sci-Fi. See Fig. 4.

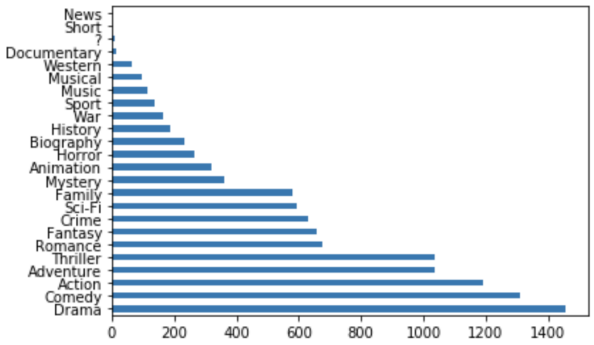

Fig. 4. Distribution of Genres

There were several Rating classifications that had so few instances associated that the classifications were combined into one classification called 'Other ratings/?' meaning that it's a conglomeration of several ratings as well as the null value type. The classifications that were merged were TV-Y7, Approved, TV-PG, TV-14, Unrated, TV-MA, NC-17, and the null type. For a visualization of the distribution of genres see Fig. 5.

It's noteworthy that Genre and Rated have a variety of possible values (for Genre: Action, Adventure, etc; for Rated: PG-13, R, etc.) with many instances for each of those possible values. However for director there is a very skewed distribution of possible values. This director appears in 589 instances out of almost 3 thousand. All other directors appear in less than 30 instances. A visualization (where outlier has been removed) is shown in Fig. 6.

To get an initial view of the relationship between features and the targets, it was useful to look at the co-variance matrix, or the column of the target in the co-variance matrix.

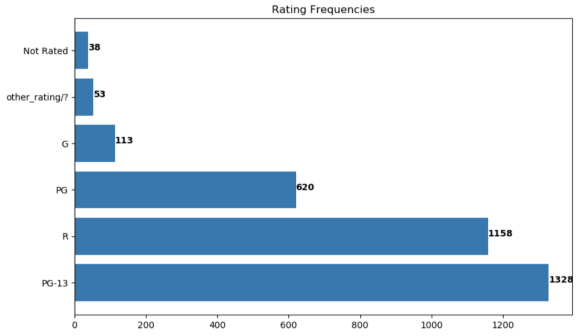

Fig. 5. Distribution of Ratings

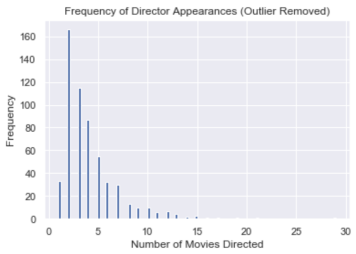

Fig. 6. Director Distribution (Outlier Removed)

Another way of understanding these relationships is through a correlation coefficient ranging from -1 to 1. Several features were notably more related to the target than other features. Some features had a negative relation (in terms of the r-value) like Director and Worldwide Gross Revenue where the r-value was -0.09.

The two most salient relationships of features to Worldwide Revenue were Production Budget, and IMDB votes. Production Budget had an r-value of .73; IMDB votes had an r-value of .59. A line of best fit, and scatter plot representation of each relationship is displayed in Figures 7 and 8.

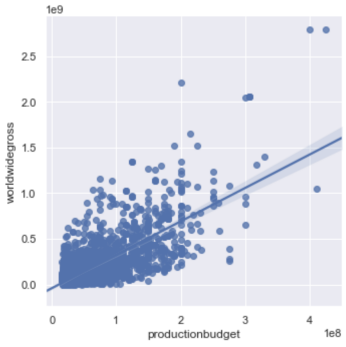

Fig. 7. Linear Fit of Production Budget to World Wide Gross Revenue

Some of the other R-values related to the Worldwide Revenue were as follows: Adventure(Genre) was 0.38, IMDB rating 0.30, and year was 0.14. Several features were strongly correlated together. Specifically MovieDB Rating and IMDB Rating were strongly correlated as well as Rotten Tomotatoes

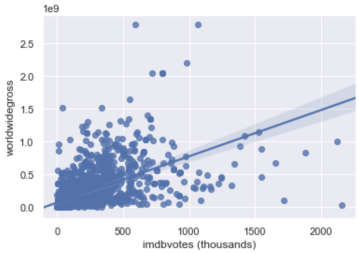

Fig. 8. Linear Fit of IMDB Votes to World Wide Gross Revenue

Rating and Metacritic Rating. Thus the number of features can be reduced by picking either the MovieDB Rating or the IMDB Rating, this reduction could be applied to the other two ratings aforementioned.

III. Results

We took some time to discuss what model would be the best to pursue to solve this problem. After some discussion we decided to try a few of them, and each took a model to try to our best ability.

A. Decision Tree & Random Forest

"The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction" (Saedsayad.com)

A Random Forest is an ensemble method meaning it uses many Decision Trees each with a different random seed in order to average the output from each tree.

A primary evaluation metric for a Decision Tree (DT) / Random Forest regression task is the explained variance score. For this score, the maximum and most preferable value is 1. The lower the score the worse it is. The default metric is R^2 score, or formally, the coefficient of determination. It also has a maximum value of 1. The R^2 score can be negative to represent an arbitrarily worse model. Both these scores were used in tandem with the Decision Tree and Random Forest.

Note that the Decision Tree & Random Forest Regressor (RFR) models initially dropped instances with the null values in any feature. This didn't invalidate the model because there were still more than 3 thousand instances even after dropping null values. Thus, there were several instances that had primarily null values with only a limited number of features with non-null values. Usually those features were the target, and production budget.

As a baseline for both metrics, the following documentation description is applicable: "A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0." This is also true for the explained variance score, and it was easily verifiable. By way of clarification, the expected value is the same as the mean.

Although some models were unrestrained in their hyper-parameters, meaning all the hyper-parameters were set to the

defaults, a Grid Search is a useful tool for experimenting with many hyper-parameter combinations. Note that scores in regards to the Grid Search are associated with a validation set that is a fold of the training data that the Grid Search is split on. The models perform worse on the testing set. The score for the Grid Search is R^2 in this case.

These models were trained to predict the Worldwide revenue as opposed to the domestic revenue.

Also note that for these models the Box Office feature was disregarded, for several reasons. First, more than a third of the instances had null values for Box Office. Second, depending upon the use case of the models this feature wouldn't be available in a production environment.

1) Initial Results: An individual Decision Tree (DT) model, that was intentionally over fit, had .999 on explained variance score. Then when the data was split into train and test sets, with a 10% split, the DT scored an .64 and with a 30% split the score was .51. These were unrestrained models in terms of hyper-parameters.

The hyper-parameters were experimented with using A Grid Search with 300 models, each tested with 3 folds of the data, meaning 900 tests. The best estimator scored a .62.

2) Tweaks and Adjustments: In order to improve the results, we attempted a Random Forest Regressor (RFR). As a baseline, and as comparison to the individual Decision Tree, the RFR was intentionally trained on all the data set, without any restrictions in the hyper-parameters. This was valuable for comparing to models that limit over-fitting.

The over fit RFR scored a .96. This over fit model didn't include null values from features. After a 30% data split for training & testing, the RFR scored a .75 with both metrics without any hyper-parameter selection.

Surprisingly, that when using a GridSearchCV to apply many different combinations of hyper-parameters the score still ranged between .6-.73. This was without imputing any missing values. The GridSearchCV primarily tested different values for max depth, the number of max leaf nodes, the minimum of leaf samples, and minimum samples for a split.

Another adjustment was to decrease the test_train split. We changed the split percentage from 30% to 10%. This change alone increased the explained variance score from .75 to .79. This was for an unrestrained RFR. In other words, all the hyper-parameters were set to default option.

Another GridSearchCV was applied and the best mean score was .73, so it appears that an unrestrained RFR performs better. Note that the same hyper-parameters were considered in the GridSearchCV.

A histogram of the mean test scores from the GridSearchCV are shown in Fig. 9.

3) Final Results: The best score was .8 on the RFR that used the default hyper-parameters. Notably this most successful model was trained on .9 of the data as opposed to .7. This score was the same for both the R^2 score as well as the explained variance score.

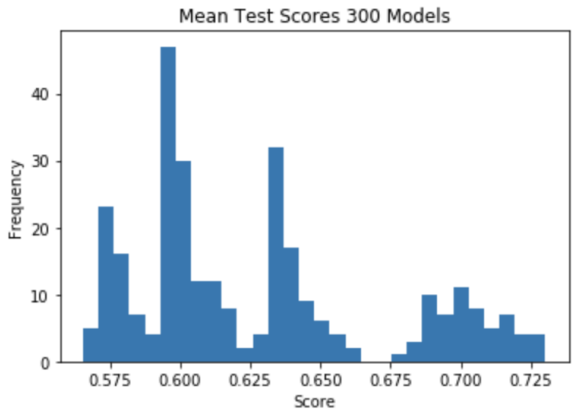

Fig. 9. Grid Search CV mean score results

With more time and computing power, a wider range of parameters could have been investigated using the GridSearchCV.

B. K-Nearest Neighbor

Another model we used is the K-Nearest Neighbor (KNN) model as defined in the sci kit learn KNeighborsRegressor class. The basis behind this algorithm is that it assumes that similar data points tend to exist in close proximity. We looked at our data and realized that most movies that have been financially successful have similar features. For example, most of them were action movies with famous directors. From that observation, we came to the conclusion that the KNN model would be a good fit for our data set. We ran each example of the model 5 times, with a different random split of the data. The numbers reported are the average of all the models.

1) Initial Results: To create a baseline to compare our progress to, we started by implementing the default model by sci kit learn. The default model uses a k value of 5, uniform weighting, and Euclidean distance. These parameters seemed like a good place to start, so we did not feel the need to change any of them for the baseline. Using this model with these default parameters, we got an R^2 value of .748 for the domestic gross, and .652 for the worldwide gross.

2) Tweaks and Adjustments: The three parameters mentioned above seemed to be the most significant, so we decided to focus mainly on them. One thing we wanted to monitor was the accuracy with regards to the k value, or the number of neighbors it takes into account when calculating the distance. Any testing that was done from this moment on was done on models with k values ranging from 2 to 100. We then chose to test distance weighting instead of uniform weighting to give more significance to the nearest neighbors. We also tested our model with Manhattan distance. We consequently found that we could fit our data better with Manhattan distance (max of .889 with Manhattan, and .816 for Euclidean), and distance weighting. Another adjustment we thought about was normalizing our features. KNN being an algorithm that depends on distances between data points, we thought we

would get better results by having all our features scaled uniformly. But surprisingly, the normalized version of our model did worse as shown by Figure 10 and Figure 11

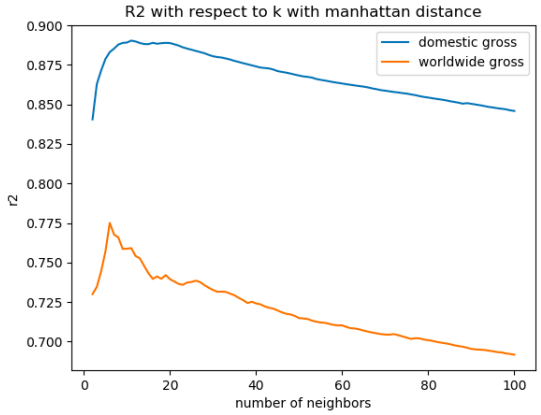

Fig. 10. Score using Manhattan distance - not normalized

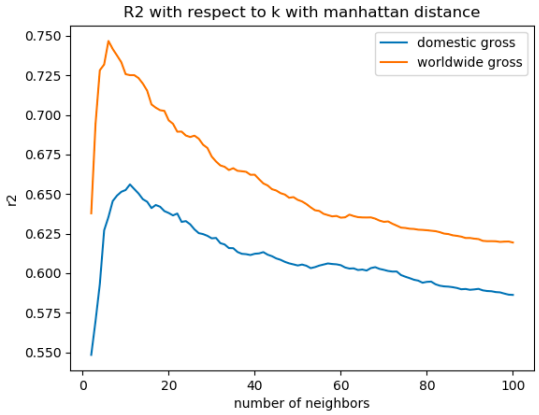

Fig. 11. Score using Manhattan distance - normalized

We also tried removing some features like 'box office' to see if it would improve our accuracy, but the initial data set ended up being the best option.

3) Final Results: After making the adjustments mentioned, the best score for domestic gross was 0.889, and 0.747 for worldwide gross. Our final model used Manhattan distance, distance weighting, a k value of 18, and features that were not normalized.

C. Back Propagation

Here we use the back propagation model as sci kit learn defines in the MLPRegressor class. When creating the results, we ran each example of the model 10 times, with a different random split of the data. The numbers reported are the average of all the models.

1) Initial Results: Using a default model from sci kit learn we hope to create a baseline to measure our progress as we begin to tweak and improve our model. Using this model, we created a baseline with an R^2 value of .7729 for the domestic data, and .7567 for the worldwide data.

2) Tweaks and Adjustments: To increase this accuracy, we decided to run the model on a bunch of different number of hidden layers, and number of nodes within the layer(s) first. After looking at other parameters, we decided this one would have the greatest impact on the model. So we began a grid search of the domestic data and created the heat map in Figure 12. We did the same grid search with the worldwide data but that graph is not included due to space requirements and because the domestic and worldwide graphs were fairly similar. The Labeled square is the highest R^2 score that we got out of all of the models. From .7729 to .851 we are already seeing a good increase in the score.

We also looked all of the other hyper parameters that we could tune within the MLPRegressor model. We ran another grid search using the optimal structure found previously on the parameters: activation, alpha, batch size, learning rate, initial learning rate, momentum and others. Most of the parameters that we tuned ended up having the default values being the best. Momentum, tolerance, and maximum iterations were the hyper parameters that were tuned the most to get our final score.

3) Final Results: After all of these adjustments, our final model scored a .8721 R^2 value on domestic data and .8182 on the worldwide data. This is a pretty decent increase in our model overall. The final model used a .85 momentum, 600 maximum iterations, and .001 tolerance.

IV. Data and Feature Improvements

Our data could have been better in a few ways. Obviously we could have collected more data samples and increased the number of data points to help prevent over fitting. The main improvements to the data was finding an optimal encoding for categories like and including genre. Since there was such variability within the number of genres and the number of possible genres it became an interesting problem. We also consider going to another website like Wikipedia to gather additional features about each movie. This could have had a great impact on the model if we were to find an additional feature that had a direct correlation to the revenue. However we did collect every feature that we thought could be useful from IMDB, and would have collected more with additional time.

One feature that we did theorize in adding was the amount of time taken to produce the movie. However we could not find a good source to discover this feature. There is always a release date with movies, but hardly a date where production began. So we ended up not being able to include this feature.

A difficult aspect that we didn't include in our models, although it was in our data set, was the Actors. Each instance did include several actors but with the limited time, and the complexity of converting this into a usable feature, we prioritized other features.

The main issue with our data was that the-numbers sorts its moving in order of highest revenue first. Meaning, that we are working with our data set containing the 3,000 highest grossing films. This non random selection can definitely skew

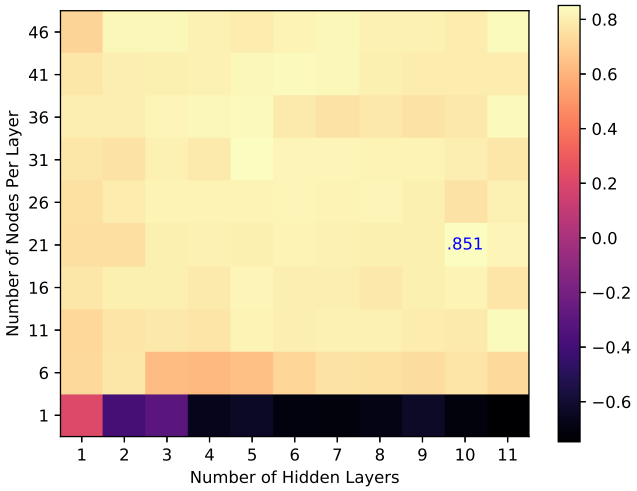

Fig. 12. Neural Network Heat map for Domestic Data

our model into thinking every movie is going to do better than it really would. As any of our models were learning, they would set the absolute minimum revenue collected to be the amount the last movie observed. We could have limited this problem by randomly picking our movies from a set of all movies. To do this though, we could have generated a random six digit number, tested to see if it was an IMDB ID, then tested to see if that movie's revenue was known. We would have had to do this a large number of times and with the OMDB API limiting our number of API calls, we would have spent a large amount of days attempting to guess correct IDs. Given more time this approach would have been viable.

V. Conclusions

KNN and back propagtion ended with results that were quite similar. After fine tuning all three models, we found that KNN did the best in the domestic category, and back propagation did the best in the worldwide category. The RF model probably struggled due to the many different hyper-parameters where adding a list of ranges for each parameter greatly increases the computational demand. With more time we could have visualized the Decision Tree, or a given Tree in the Random Forest to understand better the optimal tree structure.

We summarize the final results of each model in the following table.

Final Results in R^2		
Model	Domestic	Worldwide
Random Forest	.745	.800
KNN	.889	.816
Back Propagation	.872	.818

VI. Future Work

If we had additional time to continue working on this project, we would have liked to fix the issues that we had with our data. Mainly, the randomized gathering of data points and creating a more realistic list. We also believe that the random forest could have performed better given more time and computing power for a larger search space.

References

"https://www.saedsayad.com/decision_tree_reg.htm" - Decision Tree Regression

"https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html" - R^2 score explanation

"https://www.the-numbers.com/box-office-records/worldwide/all-movies/cumulative/all-time" - The Numbers Revenue data link

"https://www.imdb.com/interfaces/" - IMDB data download link

"https://developers.themoviedb.org/3/getting-started/introduction" - The Movie DB api link