Abstract

A new NBC affiliate wants to make movies to show on their station, so it needs to know which scripts will make well-liked movies. I used Netflix ratings, genres, and published movie scripts to make a recommendation machine for new movie scripts. To create the prediction model, I tested a gradient booster, a random forest, and linear regression. I also tested for an optimal number of features. Using root mean squared error as a metric, the linear regression model did the best at predicting ratings from scripts.

Introduction

XYZ is an NBC-affiliated station. It is new to its market. It has the stable of NBC shows behind it, but it is looking for other ways to improve its ratings in its market. Management has decided that broadcasting new movies may help. They feel that well-rated movies will improve ratings.

As an NBC affiliate, XYZ has access to some money. From this limited pot of money, they can produce their own movies. Management thinks that producing movies will not only improve their ratings, but improve their standing with NBC.

XYZ has access to scripts for the movies, but the movies do not have ratings from customers. It wants to know which movies to produce, because the money from NBC is limited. That is, XYZ wants to **predict** how well a movie will rate based on the script so they can choose highly-rated movies for production and broadcasting.

I will create a recommender system that predicts ratings from scripts. I will only use scripts and genres, because that is the data the station will be using – no cast or crew names will be involved in this recommender system.

Data

In order to create a recommendation machine that will predict how well movies will do using just scripts, we need scripts of produced movies and some form of ratings for those movies. I used https://www.simplyscripts.com/movie-scripts.html to find scripts. For movie ratings, I used information downloaded from IMDB, stored at https://www.kaggle.com/datasets/ashirwadsangwan/imdb-dataset.

The IMDB information has five .tsv files on Kaggle, but I was only interested in three of the five for the script recommender. The first file of interest contains title, type of media (e.g.: television, documentary, short), genres, and year in addition to other information; the second contains the title and language of the film in addition to other information; and the third file contains the ratings of the media. The media are linked across files by a unique code. There are over 10 million titles, and 1.4 million ratings of the various media.

The scripts were on various websites, and linked to from the simplyscripts.com website. The scripts themselves are published in one of three formats: HTML, pdf, and text (.txt). There are about 1,000 links on simplyscripts. They are all to American fiction movies.

Data Wrangling

Ratings

The IMDB data was very clean; however, there was a lot of extraneous information. In order to find the ratings for just movie scripts that are linked on simplyscripts.com, I needed to delete other media (e.g., short, documentary), movies from other countries, movies in other languages, and movies that aren't yet made.

I had three files that contain this data in separate parts. In order to delete the unnecessary data, I used Pandas to link the year dataframe and the language dataframe by their unique IMDB code in a new file. In this new dataframe, I deleted all the non-movie media (e.g. television); movies from other countries; movies in other languages; and unmade movies (as determined by year of movie).

After deleting these unneeded media, I then used Pandas to link the remaining movies to their IMDB ratings from the third file. This resulted in a dataframe that had the name of the movie, the year, the genres, and the rating.

After doing the linking and cleaning, I was left with 1100 movies and their ratings to link to the script of the movie.

Before I could link to the scripts, however, I needed to convert the genres into data a recommender system can use. I decided to convert the genres into a bag-of-words. A bag-of-words is a format where all the words in the document are represented as counts. A bag-of-words also removes the context in which the words occurred. At the end of the conversion from text to bag-of-words, the words are the features and their count is the data.

Scripts

The scripts were not as clean as the IMDB data. The scripts were not on simplyscripts.com, but rather linked to from the site. I wrote a function using Beautiful Soup to open new links, and download the script onto my computer. Some of the links were bad, such as those pointing to missing webpages, and thus had to be avoided during scraping. I also saved the name of the file in a list so I could access the files now on my computer as I was cleaning them.

The downloaded scripts were in different formats: some were webpages and in HTML; some were pdf files; and the remaining were .txt files. I would have to deal with this format difference in order to use the scripts as data.

I still had other additional steps that needed to be conducted before the scripts could be used as data. First, the simplyscripts website contained links to unmade movies. These are unusable: they won't have ratings. I deleted the unmade movie names from my list of files, so these scripts wouldn't be accessed by the functions I would write to clean and convert the scripts.

I then manually spot-checked the files that I downloaded. I found that several of the downloaded files were merely notifications that the scripts were no longer posted. I manually deleted all the files that were not scripts. However, the script name was still in my list of files, as this was a manual deletion of files, so I would have to account for the missing files as I cleaned and converted the existing files.

Having deleted files that weren't scripts, I had to convert the remaining files into a single format so one function could be used to import the scripts into Python. I transformed the list of files to three lists, each containing only one format of scripts. I wrote functions to convert the remaining files from either HTML or pdf format into .txt format, using the list of files that was appropriate to that format. Each function also skipped files that had been manually deleted. Thus, in each of the functions I wrote to convert formats, I created a new list of scripts that did not include the manually deleted scripts.

Although the .txt files were already in the appropriate format, I needed a new list that didn't contain the names of deleted .txt files. I thus wrote a third function that read through the files that were already in .txt format and saved them again as .txt files. This function also wrote the names of the existing files to a new list.

I now had three lists of existing file names: one each for originally HTML format, pdf format, and .txt format. I also had all the scripts as files on my computer in .txt format. I combined the three lists into one master list of file names; this master list only included existing scripts.

Now that all the files were readable scripts of existing movies in .txt format, I needed to write the scripts into a data file that a recommender system could use. I thus wrote a function to use the master list to read the scripts and movie names into a Pandas dataframe. The new dataframe had two columns: the name of the movie and the script of the movie.

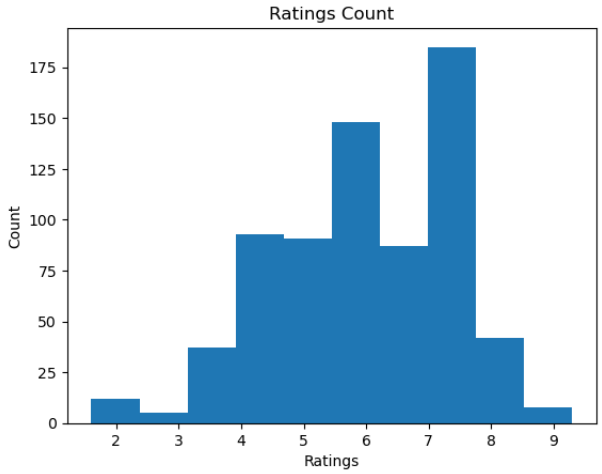
However, a recommender system cannot use this format. So my last step in cleaning and converting the scripts was to convert the scripts into a bag-of-words, as explained in the ratings section. This gave me a sparse array of over 6000 features (words) with over 700 samples (movies).

<u>Scripts and Ratings</u>

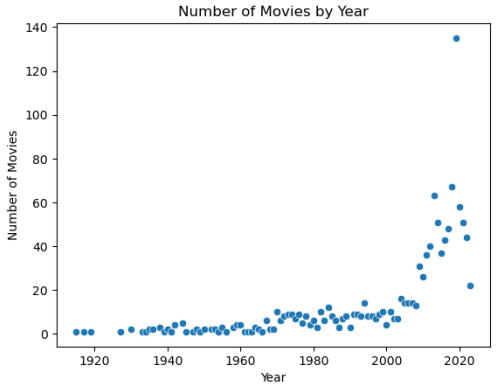
I used Pandas to merge the script dataframe and the IMDB dataframe via movie title into a final sparse dataframe that contained all the words from the scripts, the genres, and the ratings. All other information from the ratings dataframe and the script dataframe, including movie title and year, was unnecessary now, so I dropped all that information. This left a dataframe that had just over 700 scripts represented as a bag-of-words and ratings; this could be read by a prediction machine.

EDA

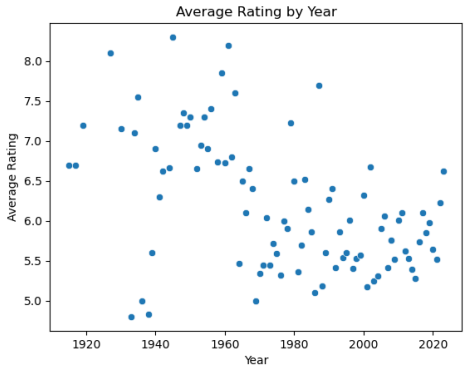
The IMDB rating system is out of ten stars. The rating of each movie is an average of all the ratings for that movie, rounded to the nearest tenth place. These average ratings for our 700+ movies run from 2.0 to 9.0 and cluster around 6.5.



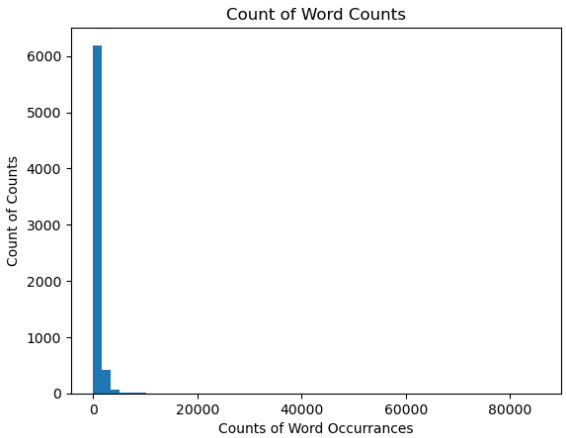
Before dropping the year, I looked at how the 700 movies were rated by year in the IMDB data, and found that most of the rated movies were from the 2000s.



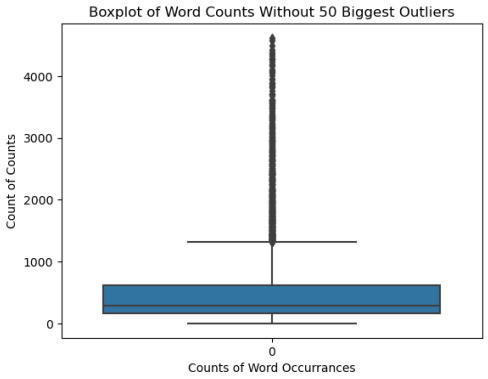
I also looked at how the ratings themselves were distributed across years, and found that the average rating for movies in a given year drops slightly as the movies get more recent. This may be a function of how many movies there were to be rated, however; recall from the graph above that there are more movies being rated in the 2000s than there are movies being rated in the 1900s.



That gave me some information about the ratings. I also had questions about the words. I looked to see how often words occurred in the dataframe. I found that, although most of the words occurred only a few times, there were a number of outliers that occurred a lot. As can be seen in the bar graph, the tallest bar is near 1. However, the biggest outlier occurs out near 80,000, meaning one word occurred over 80,000 times across all the movies.



This boxplot of word counts without the fifty highest outliers shows there are still a lot of outliers. The word counts themselves were not going to help with predicting the ratings of the movies.



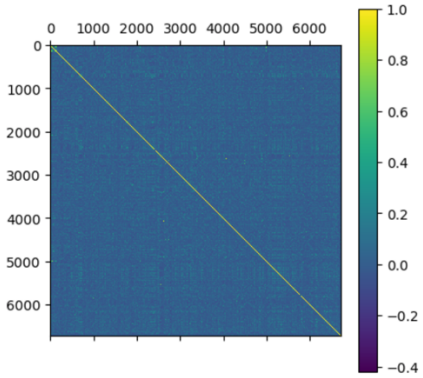
Moving to the combined data, I wondered how the individual words interacted with the ratings for the movies the words are in. I looked at the correlation between the words and the ratings. The correlation for individual words to ratings was not very high. Here is the correlation for the top 5 positively correlated words, and the top 5 negatively correlated words:

	words	correlations
6701	genre_horror	-0.27
6703	genre_musical	-0.21
6710	genre_thriller	-0.16
6476	watched	-0.14
6037	thrill	-0.14

	words	correlations
878	buildings	0.13
6689	genre_adventure	0.14
6702	genre_music	0.15
6712	genre_western	0.17
6695	genre_drama	0.22

Interestingly, the words that have the highest correlations with the ratings are genre labels.

The correlation between words was also low, as shown by the correlation matrix below. The yellow diagonal shows each word correlated with itself. Note, however, that the matrix is mostly the light blue-green indicating no correlation:



Having examined the data for information, it was time to turn to modeling.

Algorithms and ML

At this point, I had a dataframe that contains about 700 scripts of produced films and their ratings. The features for this dataframe are the individual words that make up the script and the genres. There are over 6000 of these words – that is, over 6000 features. Due to the curse of dimensionality, this is too many features for only 700 items if we want to make a good recommender. I needed to reduce the number of features. There are several possible tools that can be used to reduce the number of features. These tools include PCA and SVD. I used SVD because the feature matrix is a sparse one and SVD is recommended for sparse matrices.

These tools do not tell us how many features to reduce to. However, there are a couple of heuristics that can help. One heuristic is to use the square root of the number of items in your file. We have 700+ items in our file, and the square root of 700 is approximately 27. Thus, I chose to use SVD to convert the 6000+ features to 27 features.

Having converted the features into a reasonable number for machine learning, I turned to the predicted variable, the ratings. Two of the potential models, random forest and gradient booster, need an integer predicted variable. This was relatively straight-forward to manipulate, as IMDB only gave the tenths place for the average ratings. I converted the ratings to integers by multiplying them by 10.

I now had a dataframe that would work for machine learning. I had a reasonable number of features, and an integer predicted variable. I needed a metric to use to compare the results of the various models.

For a metric to compare the various prediction machines, I chose root mean squared error (RMSE). RMSE is a commonly used accuracy metric for values that are continuous rather than binary. In addition, another factor that makes it a good metric is that it down-weights outliers.

With the features and predicted value ready, I needed to decide what kinds of models to use. I decided to test three model types: random forest, gradient booster, and linear regression. These three are good models for continuous predicted values. Random forest and gradient booster are powerful methods for prediction. Linear regression is less complex than random forest and gradient boosting, but it has the benefit of being easily interpretable.

I also wanted a very simple model to compare these other models to. I chose the mean of the ratings as the simple model; that is, all scripts predict the mean of the ratings. This gave an RMSE of 13.923. I then turned to the more complex models.

I first tuned the random forest model to see how well it could do at using the scripts to predict the ratings. There are three main hyperparameters for a random forest model: number of trees in the forest, max-depth for how many branches the model can use, and criterion for determining if the branch should split again. Using RMSE as a metric, I tuned the hyperparameters in the random forest learner. I found that the best hyperparameters for the random forest on this dataset are:

number of trees	120
max-depth	None
criterion	'Gini'

However, even with the tuning, the RMSE was 15.156. This is higher than the simple model, indicating random forest would not be a good model for our predictor.

I next tuned the gradient booster model, which is a twist on a random forest. There are four main hyperparameters for a gradient booster. The learning rate is how fast the model converges on a solution. The number of estimators is how many trees to grow in the forest. The maximum features is how many features the model considers when deciding whether to split the tree again. Maximum depth is how big the trees should be. For this dataset, the best hyperparameters for the gradient booster are:

learning rate	0.05
number of estimators	250
max. features	10
max. depth	2

This tuned gradient booster had a RMSE of 15.761. Again, this is higher than the simple model, and indicates a gradient booster would not make a good recommender for this data set.

Linear regression does not have any hyperparameters to tune. With this dataset, linear regression gave an RMSE of 12.040, lower than the simple model and indicating linear regression might make a good recommender for this dataset.

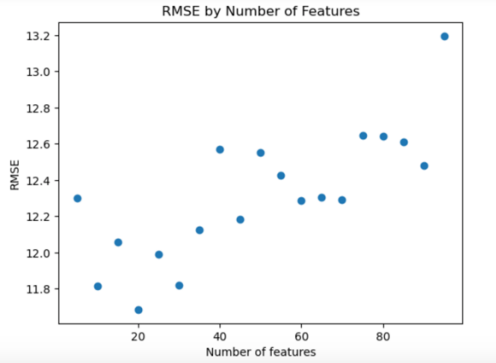
In summary, after tuning the gradient booster and the random forest, I compared the RMSE for the four models. The best RMSE for each of the four models is:

simple model (average rating)	13.923
gradient booster	15.761
random forest	15.156
linear regression	12.040

As noted above, linear regression has the best results of the four models with this dataset.

Although linear regression has the best RMSE of the four models as such, I wondered if I could get a better RMSE. With these RMSE results, I reconsidered the SVD as a feature reducer. Perhaps another format for reducing features would produce a better RMSE. Additionally, I was concerned as SVD combines multiple existing features into individual features, and these are less interpretable than the original features. I wanted to see how the original, easily interpretable features would do. However, I still had the problem of too many features for the number of instances. I thus tried the Scikit Learn method of SelectKBest, which extracts *k* of the original features to use as features in models.

I tested several values of *k* to find the ideal number of features for the linear regressor, using RMSE as the metric. Once I had a *k*, I planned to compare the RMSE result to the dataset that had 27 features created by SVD. I was focused on the linear regression as it did the best of the three models. I compared linear regressors with different numbers of *k* as selected by SelectKBest.

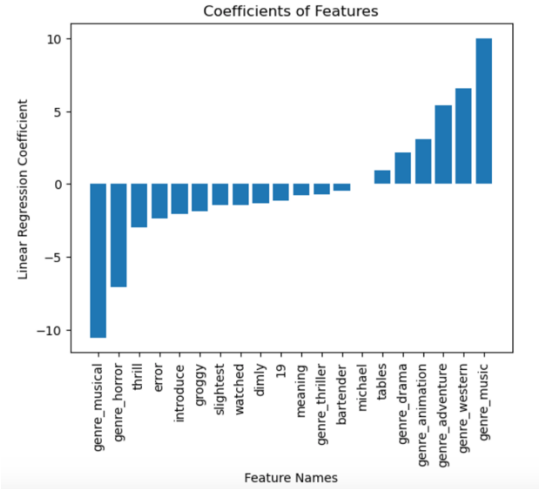


As can be seen in the plot, 20 features gave the lowest RMSE when comparing *k*-best-features in a linear regressor.

The RMSE for 20 features is 11.685, which is less than the 12.040 the linear regressor got with the 27 SVD features. So *k*-best features is a better feature reducer than the SVD for this predictor model with this dataset. Given this result, I chose the linear regressor with 20 selected features as found by SelectKBest as the recommender system.

Now that I was using original features, I could extract both the features – i.e. words – that were chosen by SelectKBest, and their coefficients as found by the linear regressor. This allows us to make some predictions about the new scripts even before we run them through the predictor machine. Coefficients far from 0 have more impact on the model. Negative coefficients are negatively correlated with the ratings; that is, a script with a high count of a word/feature that has a negative coefficient is more likely to do poorly in the ratings. Conversely, a script with a high count in a word/feature that is has a positive coefficient is more likely to do well in the ratings.

I examined the features – words – and their coefficients to see if we could use them as predictors for the dataset even before the script is run through the recommender machine.



As found in EDA with the plain correlation of words and ratings, the words that have the highest absolute values of coefficient are genres, rather than words in the scripts. For example, “genre_musical” has the highest negative coefficient, and “genre_music” has the highest positive coefficient. However, the genres are different than just correlation of words and ratings. For example, “genre_thriller” is in the top five most highly negative correlation with the ratings, but it does not appear in the feature set as determined by SelectKFeatures. We should focus on the features in the recommender machine.

Making Recommendations

Now that we have a recommender machine with easily interpreted features, we can make a prediction about how a new movie will rate. Let’s look at the coefficients more closely.

As noted above, the highest negative coefficient is for “genre_musical”, and the highest positive coefficient is for “genre_music”. However, going back to the original IMDB data and doing a spot check of movies with one or the other of these genres shows that these genres both refer to the same kind of movie – musicals. Given this contradiction, the recommender can’t say anything about whether the station should make a musical without more data.

Looking at the other coefficients, it's clear the station should not make a horror movie. Also, although animation does have a positive correlation with ratings, the cost of making an animated movie makes that genre a poor option for the station. As it is now, the station's best chance at success would be with a western or an adventure movie. The station can run the scripts for these genres of movies through the recommender to pick out the best ones to produce.

Future work

I created a recommender system for this fledgling station to use to predict which movie scripts are most likely to produce well-rated movies. However, this recommender can be improved with future work.

First, more scripts will make the script recommendation machine stronger, as more data always makes a model stronger. One possibly easy addition would be later movies: the current selection of scripts on simplyscripts.com only goes through 2009.

Other work that could refine the recommender system would be to drill down more into the "music" vs. "musical" genres. It would help to see if there exists additional information that would separate out the two genres; this way, the station could add "music" as a genre of movie scripts to check, given that it has such a high positive coefficient.

Finally, IMDB is not the only organization with movie ratings. Future work on the recommender could use other ratings organizations such as Rotten Tomatoes or Amazon, and that might produce a more robust recommendation machine.

Conclusion

A new NBC station would like to produce movies, and wants a recommender system to help them choose movies to produce. Using ratings data from IMDB and published movie scripts on the web, I created a recommender machine that can be fed just scripts and genres and predict the ratings the movie will get if it is produced. I tried several models and various forms of feature manipulation and found that a linear regressor with just 20 of the 6000+ original features produced the best predictor. Results from the model indicate that adventure movies and westerns are most likely to be the best kind of genre for the station to produce.