Chloe Zeller

Machine Learning

**Project 3: Internet Movie Database (IMDB) – Predicting Outcomes of Film Production**

**Introduction**

In this prospectus, I present a set of model that were developed using data from the Internet Movie Database (IMDB). These models have the potential to describe how length of the film’s title, or the runtime of the film, are correlated with the most success (measure in the film’s rank, rating, and revenue). They also have the potential to operate as predictors for future film data. For example, given information about an upcoming film, these models could help predict how much money the film will make, what it’s rating will likely be, and its rank.

*Data*

The data used for this project come from IMDB, a website that catalogues information about movies. The dataset had 9 predictors total:

* A short description of the film
* A list of the main actors
* A list of the genres the film falls into
* The title of the film
* The director of the film
* The year the film was made
* The runtime of the film (in minutes)
* The number of votes for the film (likely collected from IMDB votes)
* The metascore of the film (a score between 1 and 100)

And 3 response variables total:

* Rating of the film (a score from 1 to 10)
* Rank of the film (out of the total set of 100 movies)
* Revenue generated by the film (in millions)

*Process*

The first stage of the project was concerned with converting all predictors and response variables into a format that was interpretable by the modelling algorithms. This meant taking predictors that were words, and converting them into meaningful numbers. An in-depth description of how this was done is in the **Feature Descriptions and Conversion** section. Once the data had been parsed and converted, each response variable was paired with a modelling technique. The process of choosing which technique to use with each response variable is described in the **Modelling Techniques** section. Once the models had been made, I designed a small function that can take data from a not yet released film, and predict the film’s revenue. This is described further in **Trends and Takeaways**. In addition to the models, I created a script to find some general trends in the data, trends that answer questions like: *what is the average length of a top grossing film’s title?* or: *what is the average metascore of a top ranked film?* These trends and their implementation will be discussed in **Trends and Takeaways**. Before we can discuss the models, however, we need to discuss the process of converting the data into a format that could be interpreted by the modelling algorithms.

**Feature Descriptions and Conversion**

Of the nine total predictors and three response variables (mentioned above), five of the predictors and two of the response variables needed to be converted.

*Predictor Conversion*

The title, genre, description, director, and actor predictors had data that was represented in words. For the title, the simple verbatim title of the film, the genre was a list of all the genres that the film could fall into, the description was a short paragraph describing the plot, the director was simply the name of the director, and actor contained a list of all the main actors in the film.

To convert the title data into an integer that could be interpreted, I decided to simply count the number of words in the title because I had read that shorter movie titles were supposedly more “catching,” and are widely believed to be better amongst my film student friends.

Converting the description into a set of integers was the most rigorous conversion of them all. I decided to implement a bag-of-words model using bigrams. That is, I went through all of the descriptions in the dataset and counted how often two words appeared concurrently and next to one another. After I had tracked this, I then summed up the instances of each pair of words in the description over all of the descriptions. I decided to implement this conversion in python for two reasons:

* MATLAB doesn’t currently support nested maps/dictionaries. I did find a page that offered an implementation that would allow me to write nested maps[[1]](#footnote-1), but I knew that it would be much easier in python
* MATLAB doesn’t allow for dynamic memory, and I wanted to be able to add new key, value pairs during every iteration, which was very easily done in Python.

For the genre and actor predictors, I did a simplified version of what was done to convert the descriptions. I simply counted how many times that genre or actor appeared overall, and then aggregated the weights for each observation individually. Again, I implemented this in Python (for the same reasons I listed above).

Converting the director predictor was fairly simple. I created a map from the name of a unique director to the number of times that director was mentioned in the dataset, and used that integer to represent each director.

*Response Conversion*

The response conversion was very simple, and mostly involved simplifying the response variables in a way that was more beneficial to the model that I was building. I converted the rank response from a unique integer anywhere from 1 to 1000, to a class integer. There were 11 classes total (0 represented ranks 1-9, 1 for 10-19, etc.). I also converted rating (which was a number from 1 to 10), to a binary code. 0 represented a film that had a rating below 7.5, and 1 was for a film that was rated 7.5 or above. This was so that I could use logistic regression to model the rank response, and a support vector machine to model the rating response.

**Modelling Techniques**

*Logistic Regression*

Rank was most appropriately modeled by a logistic regression model because rank represented a set of 11 discrete classes, and I wanted to predict the probability that a given observation would fall into each class, selecting the highest probability as the predicted class.

I decided to train my logistic regression on all nine predictors, which resulted in a somewhat confounding solution. My predictor matrix had a very large condition number, so when I performed operations on the matrix, the operations were unstable. That said, I did have a model once I finished running my data through the algorithm. Below are the error rates for the model. The first error rate comes from doing a standard logistic regression, and the second comes from performing the logistic regression with an 8-fold cross validation.

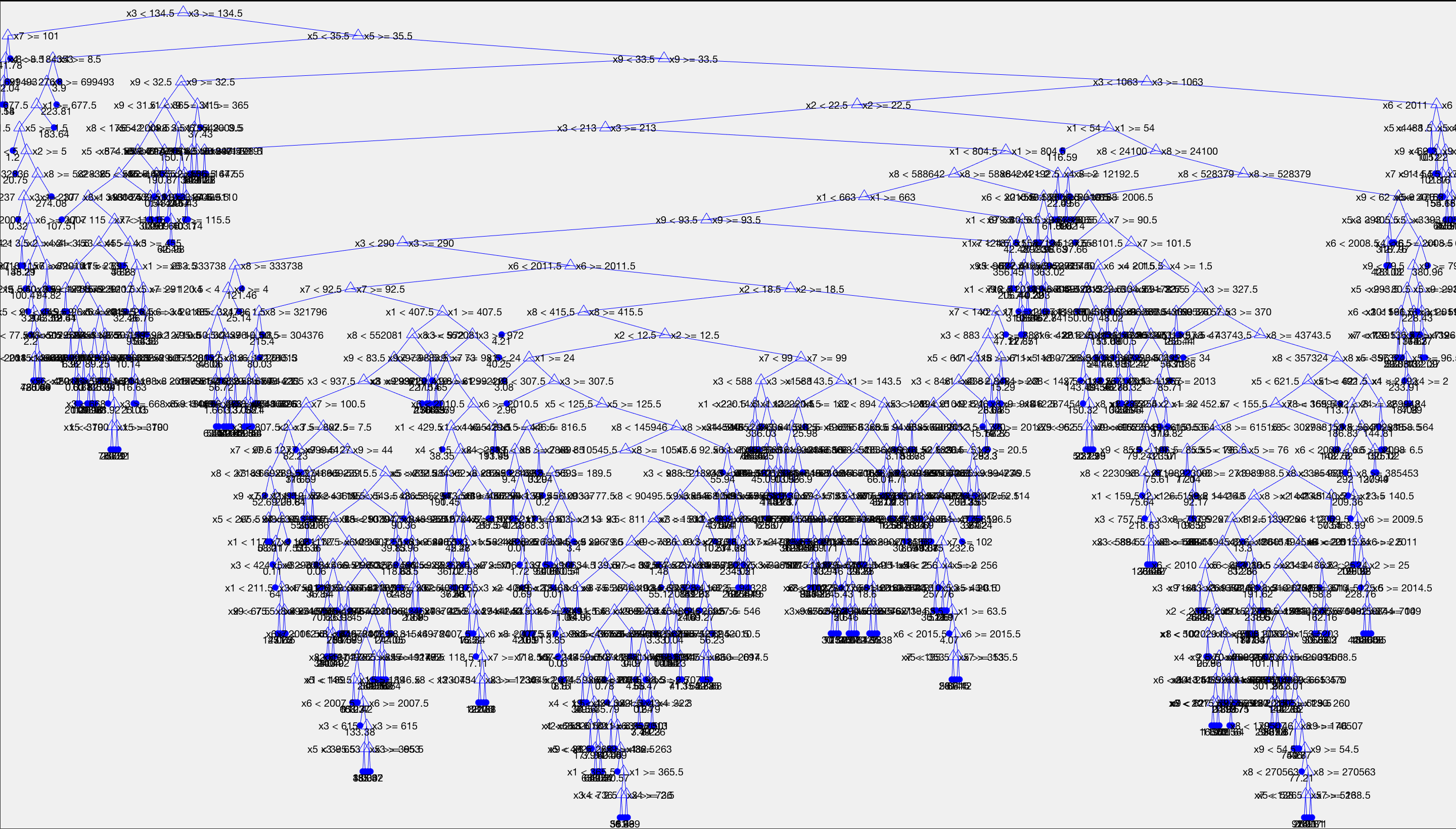
|  |  |
| --- | --- |
| **Original Logistic Regression Error** | 0.750 |
| **8 Fold Cross-Validated Logistic Regression Error** | 3.361 |

The increased error in the latter regression could be attributed to the instability in the operations performed on a matrix with such a large condition number.

*Decision Trees*

The only response that was appropriate for Decision Tree modelling was revenue because revenue was the only continuous response variable. The model was trained on the full predictor set. This was the most successful modeling technique, which could be attributed to the use of random forests to train the model (choosing a subset of features to consider at each bootstrap, then conducting majority vote). The error rate is consistently far below that of the other modeling techniques.

|  |  |
| --- | --- |
| **Random Forest Error** | 0.1273 |

** Because random forests (random forestation?) is an inherently randomized process, the error rate changes every time you run the code, however, so far in testing it has never surpassed 0.15

To the right is a sample visualization of the full tree model.

*Support Vector Machines*

Finally, in order to use support vector machines, we needed a binary response variable, and rating was easily converted to a binary variable once we had set the threshold as 7.5.

Support vector machines also performed well in terms of error rate:

|  |  |
| --- | --- |
| **Support Vector Machine Error** | 0.1583 |

and to further bolster its reliability, it was trained using cross validation (with 10 folds – the default for support vector machines).

**Trends and Takeaways**

One very interesting thing that precipitated out of having a model to describe the relationship between predictors and response, is that we can write functions that take in data that describes a yet-to-be released film, and use the model to predict the rating, rank, and revenue of the film. This would be exceptionally important to a budding film company, because it allows producers to compare possible titles, actors, and directors, and see how they might affect their revenue or rating, for example. It’s important to acknowledge a major disclaimer with this tool, however, these models are not perfect, and even if you were to use them to predict success, they might be unreliable. With the time constraints, and the complexity of each model, there was only time to put together a function that predicts expected revenue.

The below table shows some of the interesting trends found in top films. For the top 25 ranked, rated, and grossing films (respectively), this table shows the average title length of those top 25 films, the average runtime, number of votes, and average metascore. This shows that it is true that a shorter movie title does seem to have some correlation with how successful it is (but also not too short, 3 words seems just about right). Most of these statistics make intuitive sense, you would expect a metascore of above 50 to be associated with a more successful film. It’s not entirely surprising that the average metascore for the top 25 ranked films is only ≈57, ranking systems can be arbitrary, and the dataset does not come with documentation on how movies were ranked. Rating and revenue, on the other hand, seem like more objective measures of success.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Rank** | **Rating** | **Revenue** |
| **Average Title Length (# words)** | 3.08 | 2.68 | 3.52 |
| **Average Runtime (in minutes)** | 109.68 | 126.44 | 129.36 |
| **Average Number of Votes** | 114846 | 396242 | 569990 |
| **Average Metascore** | 56.6 | 73.08 | 70.2 |

**Future Work**

There are some feasible extensions to this project that could not be completed due to time limitations. The most prudent next step is to make sure the predictor set is stable enough to perform operations on, especially in the context of logistic regression. Second, once the logistic regression model is a bit more stable (and it has been confirmed that the model did in fact, converge), it would be great to expand the prediction functions to include rating and rank. To have a more rigorous set of functions to predict success would be a great asset to this project.

Finally, due to the fact that some of the predictors were converted to integers in a python script, it was difficult to make generalizations about the correlation between actors, genres, and descriptions and film success. In the future, it would be great to write a script that can essentially invert this information, and find the kinds of trends in **Trends and Takeaways** for the rest of the predictors.

1. https://www.mathworks.com/matlabcentral/fileexchange/62492-rolandritt-matlab-nestedmap [↑](#footnote-ref-1)