Machine Learning course project 1: Movie success

Valter Prykäri 35065

Motivation and goal

The success of a movie can vary on many variables. It would be interesting to gain more insight in this with the help of machine learning algorithms, as the data available is otherwise easily too overwhelming for the human mind. The goal of this project is to build two models based on machine learning techniques that are able to predict the success or failure of a movie. Success of a movie can be defined in many ways, so for this project I have decided to use the rating they have received on IMDb. In this case, a movie is considered successful if the rating is 7.5 or higher. This manuscript will give a short explanation of the project.

Approach

Various data about movies can be found via the omdb API. With the help of the API, the user can do requests based on a movie title or ID, and return the information as a JSON file. However, to be able to effectively do requests to the API, I will first need a list of all movies I want to do requests for. I therefore decided to scrape the IMDb website for all movie titles dated 2005-2015 available on IMDb. There are some restrictions on the IMDb website. The site can only view 250 titles per page, and can maximally view 10000 titles (40 pages).

For the project, I have decided to use R programming language, as it works well for data analysis and I have previous knowledge in the language. R contains a handy package for web scraping, called “XML”. As the IMDb website can only show up to 40 pages, I decided to do the scraping based on release year of the movies. This approach works, because there is less than 10000 movies released each year. Using a simple loop, the code iterates through all 40 pages and scrapes the titles. The last pages will naturally be empty but it is no problem as empty pages return nothing. After each year, I just changed the year-part in the URL and ran the loop again, adding titles to the final movie list. The total amount of movies is around 77000. I also used a VPN to get the movie titles in English instead of Finnish.

As the scraping was done, the next step was to do the actual requests to the omdb API. I wrote a loop to iterate through every title in the list. Every title is changed to the right format for the API and then a request is done to the API using the “httr” package. The data returned from the request is then saved into a data frame. This is where it gets a little more complicated. The requests are completed at a slow rate varying from 1-3 requests/ second). The request rate also slowed down over time. The next problem was interrupts, sometimes the request returned an error, causing the loop to stop. Luckily I could just continue running the loop from where it stopped. The loop stopped a few times for me, and the total request time for the training set ended up being over 24 hours. The total amount of successfully requested movies was ~61000. I used the same code (with small modifications) to extract the test set (movies for 2016). Most of the movies contained incomplete

data anyways, so a better approach could have been to restrict the requests to only the most voted (not highest rating!) to speed up this process. An improved approach would have been to utilize multithreading or splitting up the workload on many computers to get faster processing times.

The next step is to clean up the data. First of all, all data should contain the IMDb rating as that is the variable we are trying to predict, so movies without the rating will be removed. I also created a new variable based on this; if the rating is 7.5 or higher, the "successful" variable will get the value 1, otherwise 0. Many movies are listed in many genres, and to make analysis easier, I used only the first genre the movie is listed in. The same goes for actors and directors, so I have removed the other actors/directors except the first one. From the release date column, I have removed the day and the year, as it is already in another column. From the length column I have removed the string "min", so that the variable can be considered numerical. With the data cleaned, it is time to do the analysis. I decided to try to build my models based on SVM, Naïve Bayes and Logistic regression. There is a package for these methods in R, called "e1078".

Discussion

By looking at the data, there are several issues in using the variables for accurate predictions. There are extremely many levels of many of the variables. Let us take actors as an example. There are thousands of actors, which means that there are many categories under that variable. This means that it is very difficult to gain any insight from these categorical variables. Many of the predictable movies will introduce new actors, which makes it even harder to do any successful predictions.

The first problem I encountered when trying to train models with the data I had was that I did not have enough memory in my computer. This is because of the complexity of the variables and the need to use many variables. Training the algorithms required more than 5 GB of RAM. I had to get access to better hardware. Even after testing with a better computer, I was not able to run SVM or logistic regression with many features. The computations took too long, or R crashed. I had to do major changes to my initial approach.

I decided to restrict the dataset, so I scraped a new list of movies consisting of the 250 most voted movies per year. My new training set contained about 2300 movies and the test set contained around 200 movies from 2016. These smaller datasets are much easier to analyze. I tried mainly building models based on SVM and Naïve Bayes, but the predictions were not very good. One reason for this was that my dataset was quite unbalanced, and at best, my models were able to reach the same accuracy as predicting all movies as not successful. Because of this, I decided to change my approach again by making my training set more balanced.

Instead of undersampling or oversampling my already smaller training set, I decided to pick successful movies from the original, large data set. I was hoping to get better predictions with the models by adding more "Successful" movies to my training set. My new (and final) training set consisted of 1921 unsuccessful movies and 1870 successful movies. This makes it much easier to build models and actually interpret results. As most movies released are still not successful based

on my definition, the test set is still unbalanced. Simply by predicting all movies as not successful, we would have a correct prediction rate of 80%.

Starting with Naïve Bayes, the prediction accuracy actually got worse with each added feature, and as the assignment had a requirement of at least 4 features, this is not optimal. The best result with a minimum of 4 variables I came up to was an accuracy of 68% using Movie length, Genre, Director and Actor as features.

Training a model with SVM gave a result of 73,8% when using only movie length and genre as features. Using only Actor as a feature, I got up to 80% prediction rate, but the model only predicts 2 movies as successful, so it is still not useful. Using MPAA, Genre, Director and Actor as features, the model only got a prediction accuracy of 64%. Introducing Director to the previous model did not change the predictive power.

After some heavy testing with different features/models, I have concluded that the models are not useful for predicting the movies as successful/not successful based on my criteria (IMDb rating). The data is quite complex with so many levels, and it is hard for any model to find any useful information. The positive thing is that the models still provide some predictive power, as they have an accuracy higher than 60% on the unbalanced test set. High rating of a movie is probably affected by so many other factors than the features available through the omdb API. Naturally, I could have used features such as metascore rating or box office, but that would of course miss the point of predicting the IMDb rating before a movie is released. Better results might be achievable through more feature engineering. My approach has several flaws as well; one of them is that I am only using the first actor listed. It is however hard to list actors in an effective way without making model training too hard. In this project, it seems like the best model would be to predict all movies as not successful. That is of course if we consider accuracy as the main performance metric. This approach naturally means that no movie will ever be predicted as successful, so from this point of view the trained models could give some value.

About the code

I have written all the code in R and provided the code as a separate R file. You will need to install base R to be able to run the code. I can also recommend using RStudio, as it provides a better user interface. All the packages needed to run are in the code so you can install them from there by running the specific lines. You cannot run all the code at once to get results. Especially regarding the web scraping and API requests you have to change some variables manually. I have also provided the datasets separately, so you can import them with the read.csv()- lines in the code. The large (full) training set was too big to share via moodle, so you can find it through the link: https://dl.dropboxusercontent.com/u/103312038/All_extracted.csv