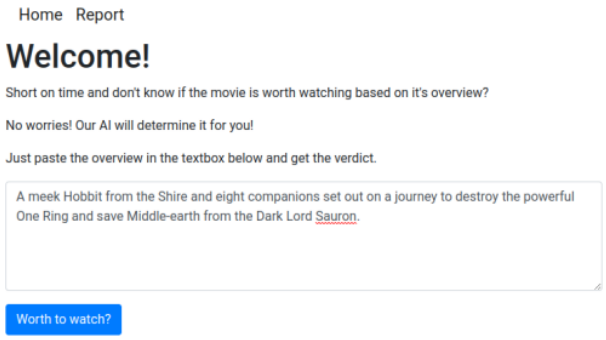
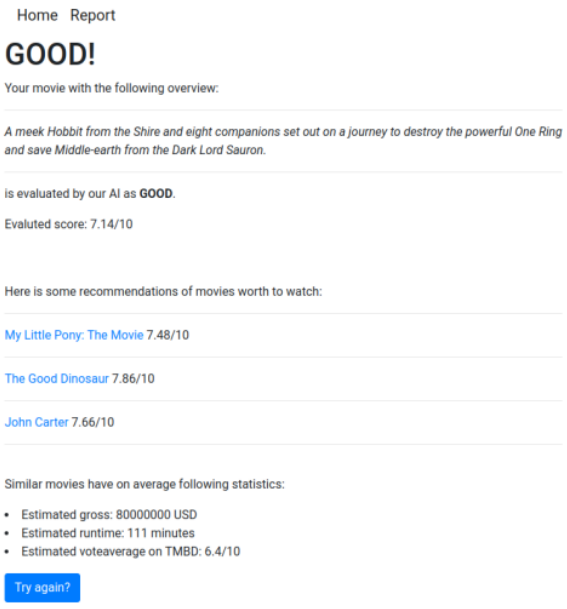
Milja Ketoja, Arttu Koskinen, Heikki Nenonen
University of Helsinki
Introduction to Data Science project

14.10.2022

Worth2Watch

– “goodness” prediction based on movie overviews





Motivation
Our goal was to find ways of determining the quality of a movie from a short description, such as a TV guide outline. This provides the user added value by giving advice if the movie is worth watching or not.

What didn't work (and why)? / Necessary changes
First we tried to use imdb data and scrape more details from another movie site, but both ideas were scrapped, as TMDB was found to be better than the (bootleg) imdb api. With TMDB, scraping was also no longer necessary. By using TMDB instead of IMDB, we were not able to access 'awards' and different critic-indexes (such as Metacritic) that were in the original plan.
Another scrapped idea was using Google Trends through the 'pytrends' package. We realized that Trends data is always relative to the specific search in question, making it less useful for comparison between different movies.
Originally we wanted only movies between a specific gross range, but through the change of the database, we decided to include everything that simply had revenue/budget data available.
Although some time was 'wasted' on the scrapped ideas, the work on them was probably necessary in order to find the right tools.

Data collecting from TMBD API
The TMBD interface required an API-key which was used as a component for the https GET request. At first we were only able to get the first 20 movies, but after realizing how to loop the pages we were able to get all the movies. We collected the data per year by ranking them in revenue order. By doing the revenue ordering we were able to limit the page requests to 50 since the following pages did not contain the required data available for our analysis. We collected data from movies over a span of years 1980-2022, each year 1000 movies, and 42000 in total. All the data was collected to a pandas dataframe.

```
url = "https://api.themoviedb.org/3/discover/movie?api_key="+ API_key
+"&primary_release_year="+ str(year)
+"&sort_by=revenue.desc&page="+str(page)
```

With the request above we were not able to get the movie genres, budget and revenue. These features were requested individually for every movie by its "id".

```
url_string =
"https://api.themoviedb.org/3/movie/"+str(movie_id)+"?api_key="+API_key
```

To reduce the amount of individual calls we preprocessed the data of 42000 movies by eliminating movies with no overview and no votes. This reduced the amount of movies to around 30000. After collecting the additional features we cleaned the data by removing all movies without revenue, budget and runtime under 60 minutes. The inflation correction was done to the budget and revenue using inflation rates from https://www.usinflationcalculator.com/inflation/historical-inflation-rates/. The final dataframe with around 10000 movies was used in the further analysis.

Overview preprocessing

After the overview texts had been collected, we turned them into several list versions that could be used when calculating the features. These included:

1. text with punctuation separated
2. tokenized
3. separated into sentences and tokenized
4. tokenized and lowercased
5. same as 4 but with stopwords removed (using NLTK's default stopword list)
6. Stemmed (we didn't end up using this)
7. With part-of-speech tags, using NLTK'S averaged perceptron POS tagger
8. Same as 7, but with stopwords removed

The objective here was to create as many versions of the overview texts as we could, that could later be useful. The POS tagger didn't seem totally accurate but still useful.

overview	overview_punct_sep	overview_tokenize	overview_sent_tokenize	overview_lower	overview_stopwords	overview_stemmed	overview_pos	overview_pos2
The epic saga continues as Luke Skywalker, in hopes of defeating the evil Galactic Empire, learns the ways of the Jedi from aging master Yoda. But Darth Vader is more determined than ever to capture Luke. Meanwhile, rebel leader Princess	['The', 'epic', 'saga', 'continues', 'as', 'Luke', 'Skywalker', ',', 'in', 'hopes', 'of', 'defeating', 'the', 'evil', 'Galactic', 'Empire', ',', 'learns', 'the', 'ways', 'of', 'the', 'Jedi', 'from', 'aging', 'master', 'Yoda', '.', 'But', 'Darth', 'Vader', 'is', 'more', 'determined', 'than', 'ever', 'to', 'capture', 'Luke', '.', 'Meanwhile', ',', 'rebel', 'leader', 'Princess', 'Leia', ',', 'cocky', 'Han', 'Solo', ',', 'Chewbacca', ',', 'and', 'droids', 'C-3PO', 'and', 'R2-D2', 'are', 'thrown', 'into', 'various', 'stages', 'of', 'capture', ',', 'betrayal', 'and', 'despair', '.']	['The', 'epic', 'saga', 'continues', 'as', 'Luke', 'Skywalker', ',', 'in', 'hopes', 'of', 'defeating', 'the', 'evil', 'Galactic', 'Empire', ',', 'learns', 'the', 'ways', 'of', 'the', 'Jedi', 'from', 'aging', 'master', 'Yoda', '.', 'But', 'Darth', 'Vader', 'is', 'more', 'determined', 'than', 'ever', 'to', 'capture', 'Luke', '.', 'Meanwhile', ',', 'rebel', 'leader', 'Princess', 'Leia', ',', 'cocky', 'Han', 'Solo', ',', 'Chewbacca', ',', 'and', 'droids', 'C-3PO', 'and', 'R2-D2', 'are', 'thrown', 'into', 'various', 'stages', 'of', 'capture', ',', 'betrayal', 'and', 'despair', '.']	[['The', 'epic', 'saga', 'continues', 'as', 'Luke', 'Skywalker', ',', 'in', 'hopes', 'of', 'defeating', 'the', 'evil', 'Galactic', 'Empire', ',', 'learns', 'the', 'ways', 'of', 'the', 'Jedi', 'from', 'aging', 'master', 'Yoda', '.'], ['But', 'Darth', 'Vader', 'is', 'more', 'determined', 'than', 'ever', 'to', 'capture', 'Luke', '.'], ['Meanwhile', ',', 'rebel', 'leader', 'Princess', 'Leia', ',', 'cocky', 'Han', 'Solo', ',', 'Chewbacca', ',', 'and', 'droids', 'C-3PO', 'and', 'R2-D2', 'are', 'thrown', 'into', 'various', 'stages', 'of', 'capture', ',', 'betrayal', 'and', 'despair', '.']]	['the', 'epic', 'saga', 'continues', 'as', 'luke', 'skywalker', ',', 'in', 'hopes', 'of', 'defeating', 'the', 'evil', 'galactic', 'empire', ',', 'learns', 'the', 'ways', 'of', 'the', 'jedi', 'from', 'aging', 'master', 'yoda', '.', 'but', 'darth', 'vader', 'is', 'more', 'determined', 'than', 'ever', 'to', 'capture', 'luke', '.', 'meanwhile', ',', 'rebel', 'leader', 'princess', 'leia', ',', 'cocky', 'han', 'solo', ',', 'chewbacca', ',', 'and', 'droids', 'c-3po', 'and', 'r2-d2', 'are', 'thrown', 'into', 'various', 'stages', 'of', 'capture', ',', 'betrayal', 'and', 'despair', '.']	['epic', 'saga', 'continues', 'luke', 'skywalker', ',', 'hopes', 'defeating', 'evil', 'galactic', 'empire', ',', 'learns', 'ways', 'jedi', 'aging', 'master', 'yoda', '.', 'darth', 'vader', 'determined', 'ever', 'capture', 'luke', '.', 'meanwhile', ',', 'rebel', 'leader', 'princess', 'leia', ',', 'cocky', 'han', 'solo', ',', 'chewbacca', ',', 'droids', 'c-3po', 'r2-d2', 'thrown', 'various', 'stages', 'capture', ',', 'betrayal', 'despair', '.']	['the', 'epic', 'saga', 'continu', 'as', 'luke', 'skywalk', ',', 'in', 'hope', 'of', 'defeat', 'the', 'evil', 'galact', 'empir', ',', 'learn', 'the', 'way', 'of', 'the', 'jedi', 'from', 'age', 'master', 'yoda', '.', 'but', 'darth', 'vader', 'is', 'more', 'determin', 'than', 'ever', 'to', 'captur', 'luke', '.', 'meanwhil', ',', 'rebel', 'leader', 'princess', 'leia', ',', 'cocki', 'han', 'solo', ',', 'chewbacca', ',', 'and', 'droid', 'c-3po', 'and', 'r2-d2', 'are', 'thrown', 'into', 'variou', 'stage', 'of', 'captur', ',', 'betray', 'and'	[('the', 'DT'), ('epic', 'NN'), ('saga', 'NN'), ('continues', 'VBZ'), ('as', 'IN'), ('luke', 'NN'), ('skywalker', 'NN'), (',', ','), ('in', 'IN'), ('hopes', 'NNS'), ('of', 'IN'), ('defeating', 'VBG'), ('the', 'DT'), ('evil', 'JJ'), ('galactic', 'JJ'), ('empire', 'NN'), (',', ','), ('learns', 'VBZ'), ('the', 'DT'), ('ways', 'NNS'), ('of', 'IN'), ('the', 'DT'), ('jedi', 'NN'), ('from', 'IN'), ('aging', 'VBG'), ('master', 'NN'), ('yoda', 'NN'), ('.', '.'), ('but', 'CC'), ('darth', 'JJ'), ('vader', 'NN'), ('is', 'VBZ'), ('more', 'RBR'), ('determined', 'VBN'), ('than', 'IN'), ('ever', 'RB'), ('to', 'TO'), ('capture', 'VB'), ('luke', 'NN'), ('.', '.'), ('meanwhile', 'RB'), (',', ','), ('rebel', 'NN'), ('leader', 'NN'), ('princess', 'NN'), ('leia', 'NN'), (',', ','), ('cocky', 'JJ'), ('han', 'NN'), ('solo', 'NN'), (',', ','), ('chewbacca', 'NN'), (',', ','), ('and', 'CC'), ('droids', 'NNS'), ('c-3po', 'VBP'), ('and', 'CC'), ('r2-d2', 'JJ'), ('are', 'VBP'), ('thrown', 'VBN'), ('into', 'IN'), ('various', 'JJ'), ('stages', 'NNS'), ('of', 'IN'), ('capture', 'NN'), (',', ','), ('betrayal', 'NN'), ('and'	[('epic', 'NN'), ('saga', 'NN'), ('continues', 'VBZ'), ('luke', 'JJ'), ('skywalker', 'NN'), (',', ','), ('hopes', 'VBZ'), ('defeating', 'VBG'), ('evil', 'JJ'), ('galactic', 'JJ'), ('empire', 'NN'), (',', ','), ('learns', 'VBZ'), ('ways', 'NNS'), ('jedi', 'NN'), ('aging', 'VBG'), ('master', 'NN'), ('yoda', 'NN'), ('.', '.'), ('darth', 'JJ'), ('vader', 'NN'), ('determined', 'VBD'), ('ever', 'RB'), ('capture',

The original overview text with the different preprocessed versions 1-8.

EDA

Before even having all the movie data collected, we tried looking for correlations between columns from a limited data set (movies only from a specific year). This was done with the help of basic python packages such as seaborn and pyplotlib. At this point, however, no interesting correlations were spotted.

Goodness factor

To decide whether a movie is ‘good’ or ‘bad’, we needed to create our own goodness factor. This was done arbitrarily by scaling four variables (gross, popularity, freshness, runtime) between 0-10, then multiplying each variable with an equal weight and finally summing results to get a grand score between 1-10.

To clarify, freshness means how new the movie is, i.e. a 2020 released movie is rated higher than a 2000 released movie. Popularity was calculated simply by multiplying vote amount with vote average. (eg. 200 votes, vote average 7 => popularity 200*7=1400).

To scale the variables, we initially used min-max-scaling. This didn’t induce enough deviation, which is why we chose to transform the data to exaggerate the standard deviation. For the transformation, we tried different things, but the easiest and most impactful was log transformation.

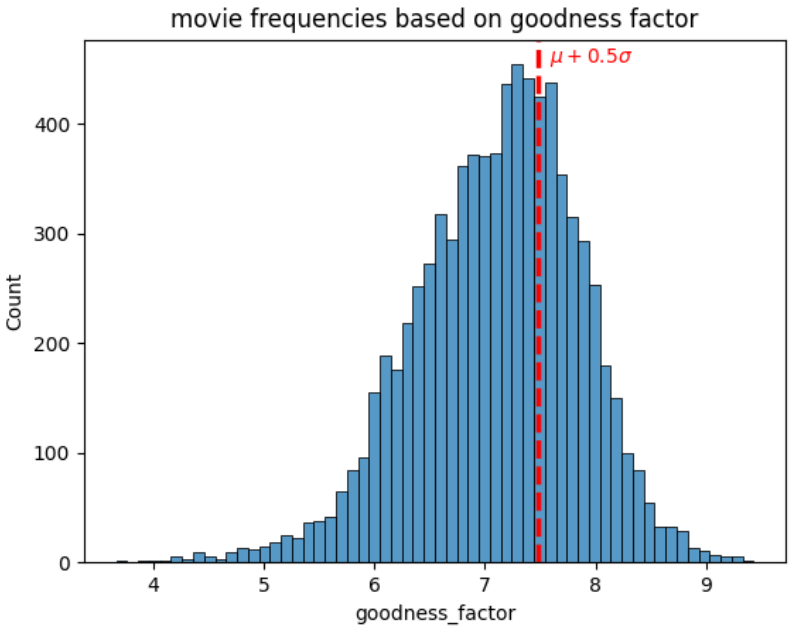
```
min_val, max_val = min(gross), max(gross)
gross = gross.apply(lambda x: 10*math.log(x-min_val)/math.log(max_val-min_val) if x-min_val>0 else 0)
```

Three of the variables were thus scaled with a combination of minmax scaling and log transformations. Runtime variable needed one extra step before scaling, which was normalizing the data, since we wanted to punish movies with big runtime deviations from the mean. Since a value close to zero was good, we then needed to inverse the data point, which was done quite arbitrarily (can be seen in the image below).

```
mean = np.mean(df['runtime'])
sd = np.std(df['runtime'])

Z = (abs((df['runtime']-mean))/sd)
inverse_Z = Z.apply(lambda x : (2*mean/(sd))/math.sqrt(x))
```

After the goodness factor was calculated, we needed to determine a threshold for good and bad movies. This was, once again, an arbitrary choice. Initially we chose a point slightly above the mean, demonstrated in the image below.



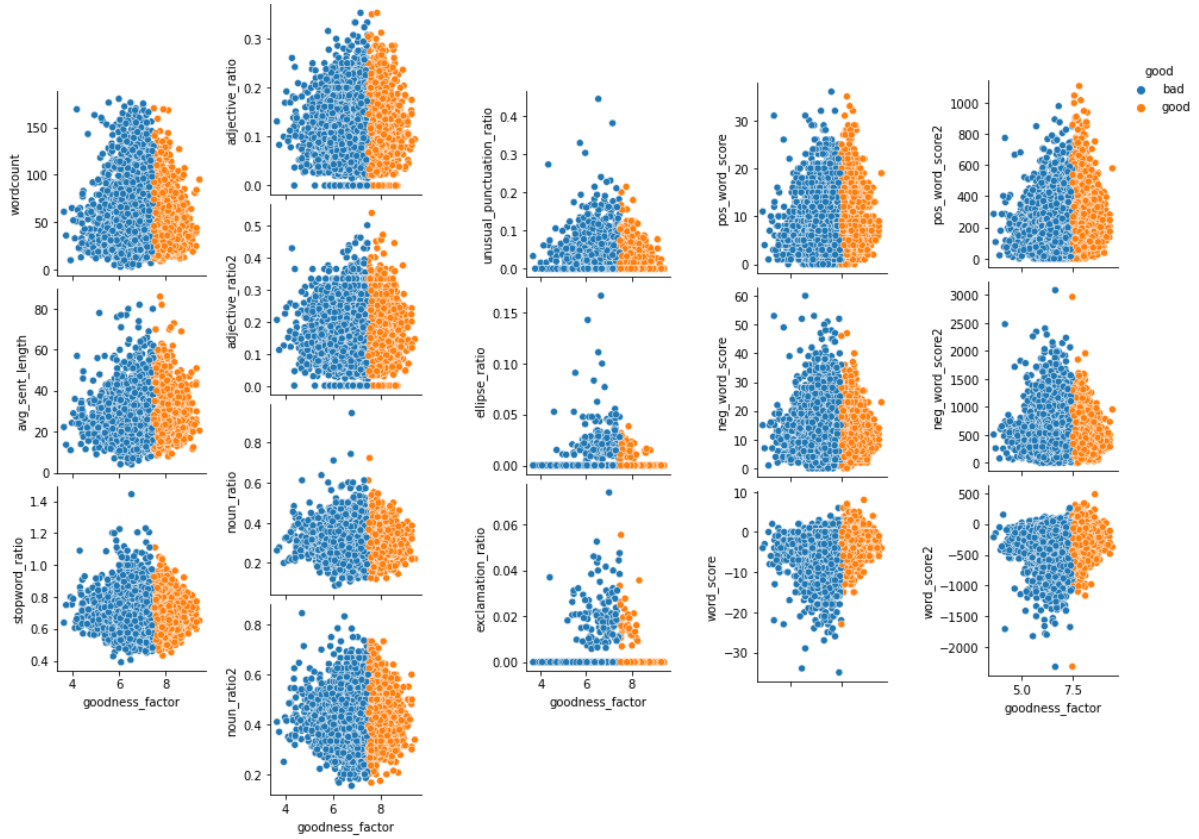
Creating text features

After preprocessing the overview texts, we turned them into sets of numerical features. The features were picked through trial and error, and using EDA we first visualized the distributions of individual features across the data. After the goodness values for the movies were calculated, we compared the features we had against the goodness factor. It was quite difficult to find features that would noticeably differentiate between the ‘good’ and ‘bad’ movies. The tested features were:

- overview word count
- average length of sentences
- ratio of non-stopwords to total word count
- ratio of adjectives and nouns (tested with stopwords removed or not)
- ratios of unusual punctuation (anything except commas and full stops)
    - ratios of exclamation points and ellipses separately
- positive/negative word score

Positive/negative word score was calculated by creating dictionaries of the most common words in each category, and deleting the top words of the other category. We tested two versions of the word score, one that simply counted word occurrences and one that weighted them using the word frequencies in the whole corpus. In the original canvas we also thought about counting instances of ‘genre words’ or doing some other analysis based on genre, but skipped that idea in the end.

The most useful features seemed to be the ratio of ellipses to word count, where high values corresponded to low goodness scores, as well as the positive-negative word score. Weighting the word score didn’t seem to make much of a difference.



Comparison of different features (y) against the goodness factor (x). The most interesting features were those that had visibly different behavior on opposite sides of the goodness scale.

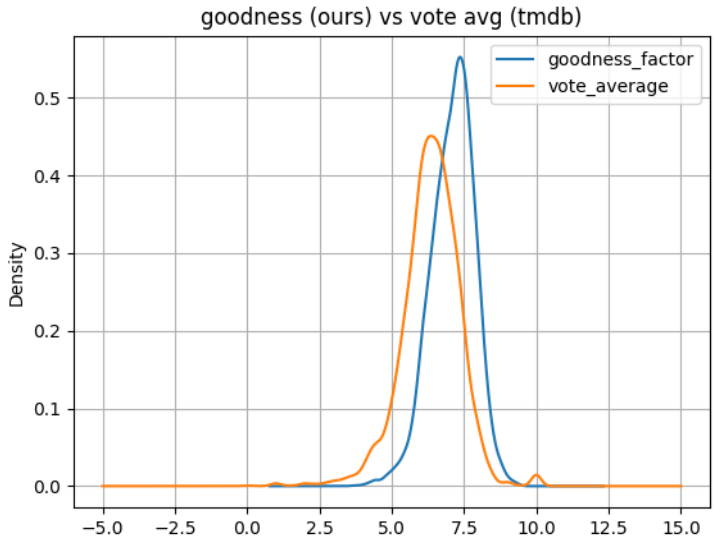
Learning task

We used linear regression to predict goodness scores based on the textual features, as well as to predict average vote count, gross revenue, and runtime. We thought about using a more complex model (for example black box neural network) but didn’t have time for it.

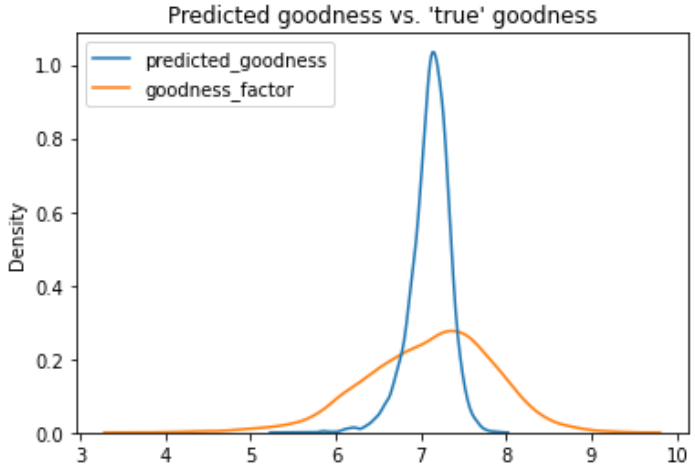
At first we tested an arbitrary goodness threshold set at the 80th percentile. This gave quite good accuracy, but the problem was that it classified almost all movies as ‘bad’. Moving the goodness factor slightly lower lowered the accuracy of the model but led to more variation in the scores. We tested different combinations of the features and tried to pick a combination that would create variation in the goodness score as well as give reasonably good accuracy, which was a difficult task since there was not that much variation in the features across goodness scores in the first place. Even though many of the features didn’t seem to

correlate that well with the goodness score, leaving only the ‘best features’ in seemed to lower the accuracy, so we used 8 features in the end. The features we used in the final model were average sentence length, stopword ratio, noun and adjective ratio, unusual punctuation/ellipsis/exclamation point ratio, as well as the word score.

Results - Analysis of the goodness factor
We compared the four factors of the goodness factor based on whether the movie was good or bad, out of which the gross factor ended up being trivial (comparison images are found in the appendix, if interested). We then compared our goodness analysis to the movies average rating (both are between 0-10):



The density plot shows that our goodness factor is slightly steeper and overall has higher values. In other words, according to our goodness factor, we rate movies slightly higher than the averages on TMDB.



We used our model to calculate predicted goodness values based on the overview text only. The model creates less variation in the goodness scores compared to the originally calculated values.

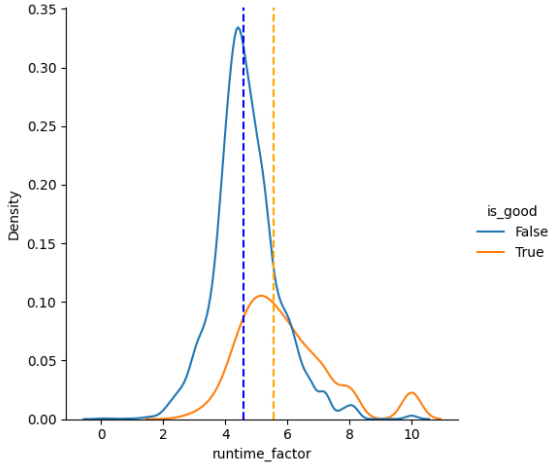
The choice of linear regression wasn't the best as the correlation between the feature sets and the different predicted values is certainly not linear. However, the main problem in the model is that there weren't that many differences between good and bad overviews we could find. Some reasons for this we could think of were the short length of the overview texts, not enough time spent on exploring different options with the features, and the way the overview texts on TMDB are created. As far as we could tell, the overview texts on TMDB are crowdsourced, which means that they are probably more even across the board than for example descriptions written by a single tv guide journalist would be.

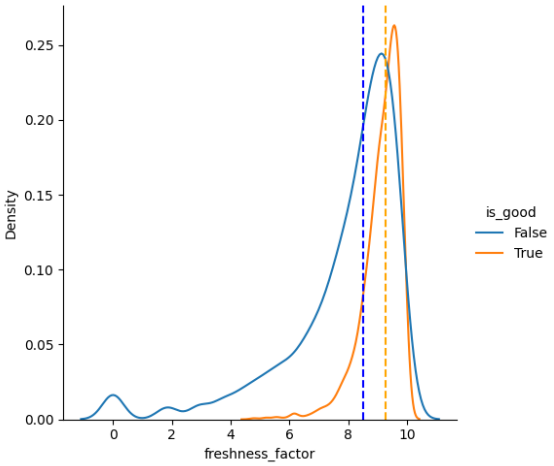
The end result

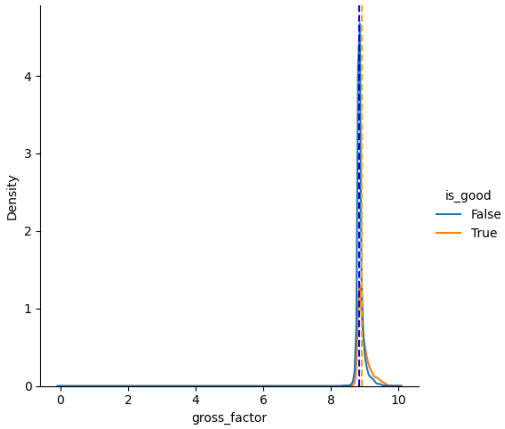
The final result was a website: https://worth2watch.herokuapp.com/. It takes the movie overview as user input in the textbox and returns a verdict from the AI if the movie is good or bad. The user is recommended three random movies that are based on our goodness factor ranked "good". Some statistics about similar movies is also provided. Screenshots from the application are given in the Appendix 3.

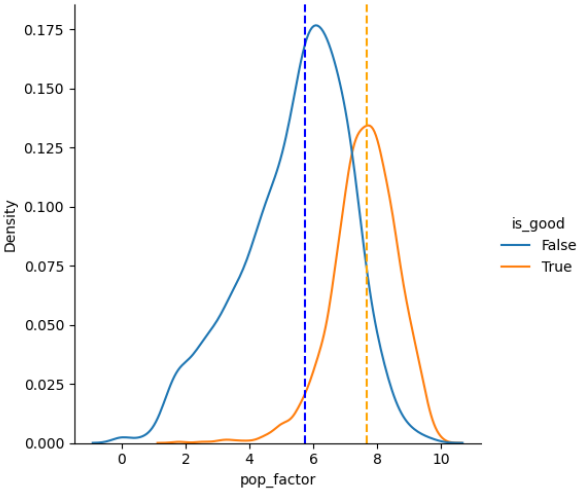
APPENDIX 1.

Factors of the goodness factor based on whether the movie is good or bad (density plots where the dotted lines indicate means):





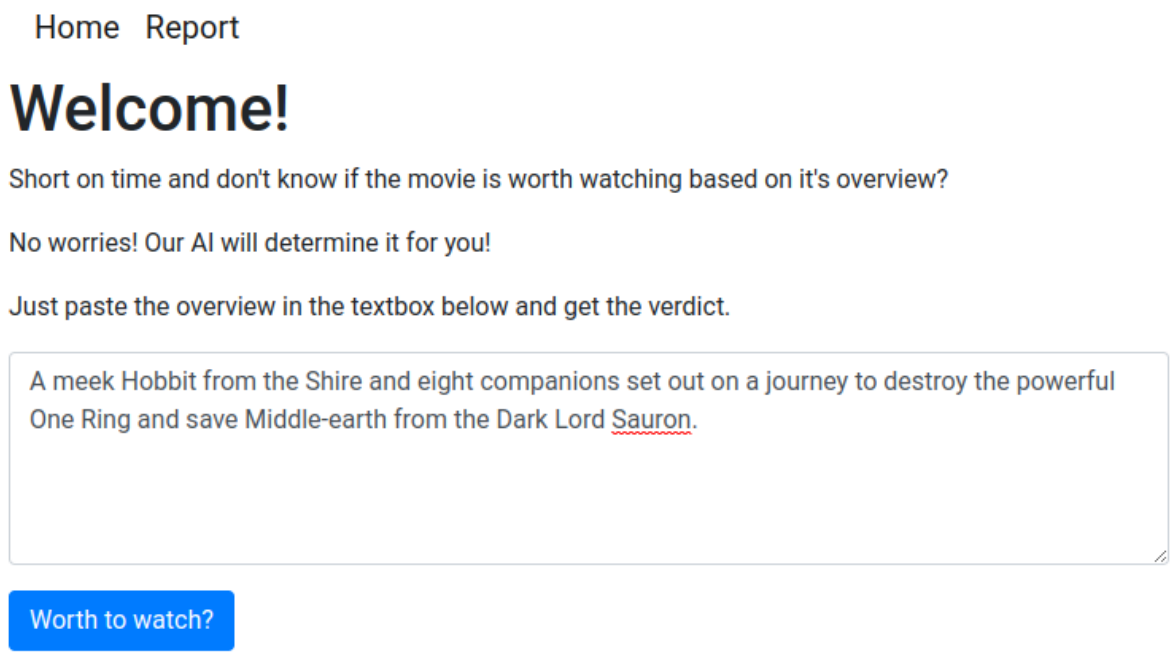




APPENDIX 2. Final form of the dataframe:

original_title	id	runtime	genres	vote_average	vote_count	popularity	year	month	day	fixed_gross	gross_factor	pop_factor	freshness_factor	runtime_factor	goodness_factor
The Empire Strikes Back	1891	124	Adventure,Action,Fiction	8.4	14567	26.960	1980	5	20	1.927225e+09	9.766115	9.363771	0.0	4.822816	5.988176
Superman II	8536	127	Fiction,Action,Adventure	6.7	1874	16.237	1980	12	4	5.053547e+08	9.299370	7.543885	0.0	4.570040	5.353324
The Blues Brothers	525	133	Music,Comedy,Action,Crime	7.8	3498	12.911	1980	6	16	3.267464e+08	9.183290	8.164259	0.0	4.161755	5.377326
Nine to Five	19494	110	Comedy	6.9	380	8.547	1980	12	18	3.455228e+08	9.196987	6.291961	0.0	10.000000	6.372237
Stir Crazy	21629	111	Comedy	6.5	345	11.162	1980	12	12	3.381161e+08	9.191633	6.166992	0.0	7.781389	5.785003
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

APPENDIX 3. App website, index.



APPENDIX 4. App website, evaluate.