Predicting Movie Success Using Text Complexity Measures

Filippo Merlo and Matteo Scianna

February 2023

Abstract

As already shown in a previous study [9], feeding Deep Learning models with linguistic information extracted from movie scripts dramatically increases the predictive accuracy of the model on the box office revenues and critical review scores, with respect to predictions based on structured data only. In our study, we ask if providing the model with more elaborated information (and thus presumably more informative) would furtherly increase the model's performance. Our approach is to use text complexity features extracted with Natural Language Processing (NLP) techniques in addition to already used successful features such as the world frequency.

1 Related work

As already said, the work that inspired our project comes from Joshua A. Gross, William C Roberson and J Bay Foley-Cox study "Film Success Prediction Using NLP Techniques" [9]. We had at disposal the Deep Learning model they used and we adapted for our purposes. The main difference between this study and our is the variety of NLP information extracted from the movie scripts and fed to the model. In our case, in addition to the word frequency feature already present in the previous work, text complexity measures are provided to the model. We do not use the words vector taken from the scene description, since it was proven that this approach had limited success.

2 Data collection

The data we used to predict the movie's gross revenue and ratings is divided into a set of structured data such as film genres (categorical), film's budget, runtimes and release year (numerical) and a set of text complexity indexes measuring the lexical, dependency and sentence complexity of the text. For the latter, we used the natural language processing methods previously used in the work [7] by Emanuele Castanò, where these complexity measures were used to differentiate literary fiction from the popular one.

2.1 Text Complexity Features

The first step was to obtain the movie script dataset from which to extract the indexes to feed the model. We modified a Python code that uses the HTML parser provided by the Python library BeautifulSoup to get the script text of 1105 movies from the Internet Movie Script Database (IMSDb) [5]. We kept the '$< b >$, '$< /b >$' HTML tags and we used them to divide the scene descriptions and dialogues (outside the tags) we needed from the heading and scene direction scripts, such as indications about how to shoot the scene or the name of the involved characters (inside the tags), we didn't use. The scripts have been further processed to keep only words, numbers and punctuation. We also applied an outlier exclusion method based on the total character number of the scripts, removing from the dataset the scripts with fewer characters of the first quartile minus 1.5 the interquartile distance (3° quartile - 1° quartile). After this exclusion step, we ended up with 1051 scripts. We then proceeded to the extraction of the NLP features from the cleaned scripts.

2.1.1 Lexical Complexity Measures

The indexes measuring the lexical complexity of the text consist of: a vector of the top 250 most common words in the script obtained with frequency analysis, and a series of lexical complexity measures

obtained with the same methods described in Emanuele Castano's work [7]. Lexical complexity is defined as the amount of variation in the lexical selection in the vocabulary of a text [7]. One index is the Type-token ratio (TTR) score 1, which is obtained by dividing the number of types of words in a text V by the total number of words in a windowed text N. The score is computed on slices of 500 words of the original text to avoid the length of the text bias (higher text length tends to increase the variety of different words).

$$\frac{V}{N} \tag{1}$$

The final lexical complexity score for the single movie script is obtained by averaging the value of every portion. We obtain nineteen different lexical complexity measures that are variations based on the TTR score. A full list of these measures is provided in the Appendix of this article.

2.1.2 Dependency-Based Measures

A series of dependency-based indexes measuring the syntactic complexity of the text were selected. Syntactic complexity is defined as the depth and length of syntactic structures of a text [7]. To compute these measures a dependencies grammar is used to establish the syntactic relationships between the words in a sentence as shown in Figure 1. From these relationships, some complexity measures such as Average Dependency Distance (ADD) and Closeness Centralization (CC) are computed. ADD represents the average distance in a sentence between the "heads" (i.e.: the words that govern a syntactic dependence) that it contains and their dependent words. CC is the estimate of the number of words that occur between a given word and its main verb in a sentence. The full list of dependency-based measures is available in the Appendix.

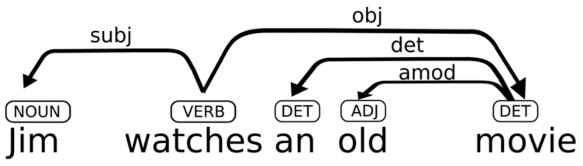

FIGURE 1: *Grammar structure example. Adapted from [7].*

2.1.3 Sentence-Based Measures

We also computed a series of sentence-based measures such as the average sentence length, calculated with both the number of words and of characters, the average punctuation per sentence and the punctuation per token index, which measures the average punctuation/words ratio per sentence.

2.1.4 Implementation

In order to extract these measures from the cleaned movie scripts we used two Python modules: Stanza [11] and TextComplexity [4]. Stanza, for which several robust language models are available, is used for a further pre-processing part. The cleaned movie scripts are converted into CoNLL-U format [2]. CoNLL-U is a version of the CoNLL (Conference on Computational Natural Language Learning) format that is a standardized text format for annotating and exchanging linguistic data. We obtained a CoNLL-U file for each movie script. Inside one of these files, each sentence consists of one or more word lines, and each word line contains the annotations for that word in 10 fields separated by a single tab character. Void lines separate one sentence from the next. A commented example of CoNLL-U sentence annotation is shown in Figure 2. Finally, the complexity indexes are computed using the functions provided by the python module TextComplexity [4] that implements various measures that assess the linguistic and stylistic complexity of (literary) texts. The Lexical Complexity Measures

```
1   They     they    PRON   PRP   Case=Nom|Number=Plur                2   nsubj   2:nsubj|4:nsubj   _
2   buy      buy     VERB   VBP   Number=Plur|Person=3|Tense=Pres     0   root    0:root            _
3   and      and     CONJ   CC    _                                  4   cc      4:cc              _
4   sell     sell    VERB   VBP   Number=Plur|Person=3|Tense=Pres     2   conj    0:root|2:conj     _
5   books    book    NOUN   NNS   Number=Plur                        2   obj     2:obj|4:obj       SpaceAfter=No
6   .        .       PUNCT  .     _                                  2   punct   2:punct           _
```

FIGURE 2: *An example of CoNLL-U annotated sentence. The ten field contain: (1) ID: Word index, integer starting at 1 for each new sentence. (2) FORM: Word form or punctuation symbol. (3) LEMMA: Lemma or stem of word form. (4) UPOS: Universal part-of-speech tag. (5) XPOS: Language-specific part-of-speech tag; underscore if not available. (6) FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available. (7) HEAD: Head of the current word, which is either a value of ID or zero. (8) DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one. (9) DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs. (10) MISC: Any other annotation. Adapted from [2]*

are computed directly from the movie script texts, while the dependency-based and sentence-based measures are computed from the CoNLL-U files.

2.2 Structured Data

To obtain the structured data we used Cinemagoer [1], a Python package for retrieving and managing the data of the "IMDb" movie database. First, we obtained the movie ID from the names of the scripts in our collection, then we used the ID to retrieve information such as film genres (categorical), film's gross revenue, rating, budget, run-time and release year (numerical). Unfortunately, this set of data was not available for all the movies, so we proceeded by manually inserting the missing data taking them from the "IMDb" website [3] and the "The Numbers" website [6]. For the model training, we finally excluded the movie with an incomplete set of data, for which the features were not available in none of the previously mentioned sources, ending with a total number of samples of 973 scripts.

3 Model

The general model structure is based on the one in [9], nonetheless a few changes have been made. In predicting the logarithm[1] of the movie revenue and the movie rating, it has been chosen to make the choice of predictors customizable, in order to test the model with different combinations of predictors. For the model building, Tensorflow [10] and Keras Functional API [8] were used. Then, many adaptations were made in order to make the model suitable for our problem. First of all, since the number of total scripts processed and the number of features taken into consideration are different to the ones in [9], the model's inner structure presents some differences in the number of layers and layers size. Furthermore, we also enabled the possibility to give as input to the model the complexity measures we exposed in the previous section.
In the end, features are provided as follows:

- Categorical features (movie genre) are supplied, for each movie, as a vector of 32 elements (the total number of genres present in our database) being 0 or 1 based on whether the genre is part of that given movie genres.
- Numerical features (Budget, Year and run-time) are given directly as input.
- For each of the complexity measures defined in the previous section (lexical, dependency and sentence-base), a vector is created and given as input.
- Similarly as what is done in [9], top250-word lists are passed through a 100 dimensions GloVe.6B word embedding and further passed through two dense layers to reduce dimensions.

Finally, Adam optimization model is used for optimization purposes, together with a mean squared error loss function. A graphical representation of the whole model can be seen in Figure 3.

[1] As exposed in [9], considering the logarithm of revenues, due to the non-linearity of the variable, enables to improve results.

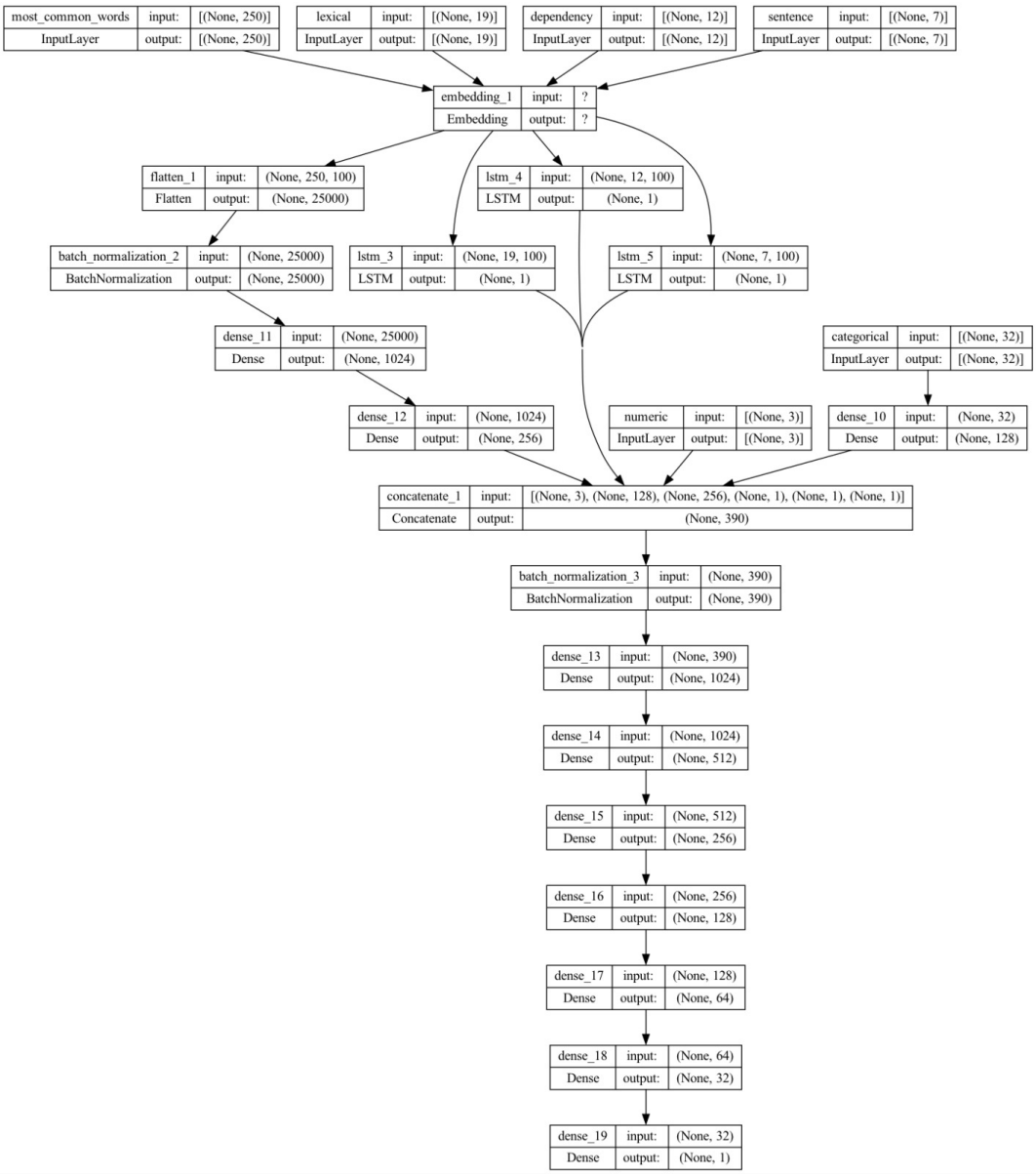

FIGURE 3: *Structure of the model (all predictors)*

4 Results and Discussion

In this section, the main results are summarized. We took into consideration four different models and compared them to a baseline. The baseline model just guesses the mean value for each of our two variables to be predicted. Other models are created by considering all or just a subset of the total features defined in the previous section. Results are highlighted in Table 1. All models except for those marked with "*" are trained for 2000 epochs, while the marked ones are trained for 4000 epochs. We noticed that the loading and handling of the frequency data (top250words) required lots of time and was quite computationally expensive, so running models containing these features for 4000 epochs would have taken too long. Even though 4000 was the epoch number Gross et. al used in [9] and the one that gave us the better results as shown in Table 1 in the marked results, we decide to run all the models with 2000 epochs and compare this results. Most of our models outperform the results of the baseline and the ones trained with 4000 epochs are quite close to the ones proposed in [9]. Looking at the 2000 epochs models we can see that the better scores have been obtained by the one using both the word frequency and the text complexity measures (log revenue MSE: 14.4226, rating MSE: 0.8302), which scores better than the one exploiting the word frequency only (log revenue MSE: 14.3129, rating

Model	log revenue MSE	rating MSE
Baseline	23.4595	0.9596
Numerical, Categorical	14.7788 (16.2193*)	0.9149 (0.7434*)
Numerical, Categorical, Lexical, Dependency, Sentence	17.8187 (14.8774*)	0.9729 (0.6228*)
Numerical, Categorical, Frequency	14.3129	0.9166
Numerical, Categorical, Lexical, Dependency, Sentence, Frequency	14.4226 (14.7066*)	0.8302

TABLE 1: Variation of Mean squared error of the test set for both variables.The results marked with * have been obtained with a 4000 epochs training.

MSE: 0.9166), the one that obtained the best result in [9]. For this reason, we can infer that the text complexity measures we introduced did have a significant impact on increasing the performance of the model. Therefore, we believe that future work into finding the model's best hyperparameter set could result in a better prediction of movie success performance than the one attained by using word frequency data as the sole movie script-based metric in [9].

Another thing we can notice from the results is a strong unbalance in MSE between the two variables, log_revenue being much more difficult to predict than rating. There are many possible reasons behind this phenomenon. In particular, revenue data often appears rounded to the hundred thousand and sometimes to the million, making it difficult for the model to correctly learn and predict a specific value. Furthermore, for some movies, the revenue value is extremely low (and sometimes even 0), due probably to a lack of information regarding the actual real value. Even knowing that those movies could lead to some unbalance in the prediction, we decided to keep those movies inside the dataset for two main reasons. First of all, since the dataset at our disposal is already really limited, we did not want to deprive ourselves of potentially useful data. Furthermore, those movies were indeed perfectly suited for the rating prediction, hence it would have been pointless to completely remove them from the dataset.

To conclude, it is important to underline that this is only the first step in order to gain confidence with these tools. The model can be improved, by finding the correct number of epochs and tuning properly all others hyperparameters. For all other considerations, we refer you to the "Further Improvements" paragraph.

5 Conclusion

In conclusion, this work aims to predict movie success in terms of the movie's gross revenue and ratings using the same model and data presented in [9] but introducing a series of text complexity measures previously used in [7]. Our findings suggest that these measures may improve the predictive efficacy of the model, but due to circumstantial complications we encountered such as a high computational demand to perform the training and incompleteness of the dataset, further work needs to be done in order to definitely address this task. Future improvements we think may be crucial are the improvement of the dataset quality, which is not enough heterogeneous since data are not always specific (especially budget and revenues that are often rounded). Some data were also missing, for example, 53 movies have a gross income of 0, hence this may have misled the metrics a bit. Better parsing of the different parts of a script would be necessary, in order to differentiate better between dialogues and scene descriptions, that in our script dataset were kept together. Finally, a deeper hyperparameter tuning would help to find the right number of epochs and generally the best set of hyperparameters for the model. The Python code containing the model and part of the NLP pipeline we used to get the scripts and extract the measures is available at the link below[2].

[2]GitHub repository at: https://github.com/MatteoScianna/predict-movie-ratings

References

[1] *"Cinemagoer"*. URL: cinemagoer.github.io.

[2] *"CoNLL-U"*. URL: https://universaldependencies.org/format.html#conll-u-format/.

[3] *"IMDb"*. URL: www.imdb.com.

[4] *"Text Complexity"*. URL: github.com/tsproisl/textcomplexity.

[5] *"The Internet Movie Script Database (IMSDb)." The Internet Movie Script Database (IMSDb)*. URL: www.imsdb.com.

[6] *"The Numbers"*. URL: www.the-numbers.com.

[7] E. Castano et al. "On the Complexity of Literary and Popular Fiction (Under revision)". In: (Under revision).

[8] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[9] Joshua A Gross, William C Roberson, and J Bay Foley-Cox. "CS 230: Film Success Prediction Using NLP Techniques (2021)". In: (Under revision).

[10] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[11] Peng Qi et al. "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages (2020)". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Online: Association for Computational Linguistics, 2020, pp. 101–108. DOI: 10.18653/v1/2020.acl-demos.14. URL: https://www.aclweb.org/anthology/2020.acl-demos.14 (visited on 02/05/2023).

Appendix

N**:** Text length, i.e. number of tokens

$V(N)$**:** Vocabulary size, i.e. number of types

$V(i, N)$**:** Number of types occurring i times

Lexical Complexity Measures

$$\text{type-token ratio} = \frac{V(N)}{N}$$

$$\text{Guiraud's } R = \frac{V(N)}{\sqrt{N}}$$

$$\text{Herdan's } C = \frac{\log(V(N))}{\log(N)}$$

$$\text{Dugast's } k = \frac{\log(V(N))}{\log(\log(N))}$$

$$\text{Maas' } a^2 = a\log(N) - \log(V(N))\log(N)^2$$

$$\text{Dugast's } U = \frac{\log(N)^2}{\log(N) - \log(V(N))}$$

$$\text{Tuldava's } LN = \frac{1 - V(N)^2}{V(N)^2 \log(N)}$$

$$\text{Brunet's } W = N^{V(N)^{-a}} \text{ with } a = -0.172$$

$$\text{Carroll's } CTTR = \frac{V(N)}{\sqrt{2N}}$$

$$\text{Summer's } S = \frac{\log(\log(V(N)))}{\log(\log(N))}$$

$$\text{Honoré's } H = 100\frac{\log(N)}{1 - \frac{V(1,N)}{V(N)}}$$

$$\text{Sichel's } S = \frac{V(2,N)}{V(N)}$$

$$\text{Michéa's } M = \frac{V(N)}{V(2,N)}$$

$$\text{Entropy} = \sum_{i=1}^{N} V(i,N)\left(-\log(\frac{i}{N})\right)\frac{i}{N}$$

$$\text{Yule's } K = 10^4\left(-\frac{1}{N} + \sum_{i=1}^{N} V(i,N)\left(\frac{i}{N}\right)^2\right)$$

$$\text{Simpson's } D = \sum_{i=1}^{V(N)} V(i,N)\frac{i}{N}\frac{i-1}{N-1}$$

$$\text{Herdan's } V_m = \sqrt{-\frac{1}{V(N)} + \sum_{i=1}^{V(N)} V(i,N)\left(\frac{i}{N}\right)^2}$$

$$\text{McCarthy and Jarvis' } \textit{HD-D} = \sum_{i=1}^{V(N)} \frac{1}{42}\left(1 - \frac{\binom{i}{0}\binom{N-V(i,N)}{42-0}}{\binom{N}{42}}\right) = \sum_{i=1}^{V(N)} \frac{1}{42}\left(1 - \frac{\binom{N-V(i,N)}{42}}{\binom{N}{42}}\right)$$

The only measure that cannot be expressed with a formula is the Measure of Textual Lexical Diversity (MTLD). The procedure to calculate this index consists in incrementing by 1 a counter every time the type-token ratio of a text reaches the threshold of 0.72. Once the threshold is reached, the type-token ratio is reset to zero and the procedure starts again. MTLD is the mean value of the counter from the beginning to the end of the text and the counter fromour the end to the beginning.

Syntactic Complexity Measures

- Average Dependency Distance; it is the mean linear distance between the words in a sentence and their heads (e.g: "Sarah(1) reads(2) books(3)", where both (1) and (3) depend on the head (2) has an average distance of $(1 + 1)/2 = 1$).
- Closeness centrality; it is defined as the reciprocal of the sum of the length of a path from one vertex to another.
- Closeness centralization; in a graph-based representation of a series of sentences, it is the average value of the sum of the differences between the highest centralization values and all the other centralization values, divided by the length of the graph, minus 1.
- Outdegree Centralization; in a graph-based representation of a series of sentences, it is the average value of the sum of the differences between the highest number of edges pointing out of a certain node, minus every other value of outgoing edges of every node, divided by a weighted length of the node. It reflects the distribution of the number of dependencies in a sentence.
- Longest shortest path; longest shortest path from the root vertex, in other words the depth of the tree.
- Dependents per word; the average measure of dependencies per word in a text.