UNIVERSITÀ DI PISA

Master Degree in Computer Science

Spoiler detection in IMDB database

Human Language Technologies final report

Francesco Botrugno — Master Degree in Computer Science
Agnese Camici — Master Degree in Digital Humanities
Francesco Caprari — Master Degree in Computer Science

2023/2024

Contents

1 Introduction

User reviews are a fundamental pillar in guiding our entertainment consumption decisions. However, the presence of spoilers can compromise the user experience, undermining the thrill of uncertainty and discovery that makes watching movies and series enjoyable. This phenomenon can generate frustration and can discourage users from consulting other people's opinion, limiting their ability to make conscious and satisfying choices.

IMDb (Internet Movie Database) is a popular resource for choosing which content to watch, offering user reviews, ratings and detailed information about movies and TV shows. By exploring ratings and reviews, viewers can make informed decisions about what content aligns with their preferences and interests. The platform offers both critical and user ratings: when writing a review, users can mark certain parts of their text as spoilers. This is done by enclosing the spoiler content within designated spoiler tags. This helps ensure that critical plot details or twists are not immediately visible to those users who don't want to see them. Reviews that contain spoiler tags are usually marked with a warning. This alert appears before the review text, allowing users to decide whether they want to proceed and read the review or skip it to avoid spoilers. However, unfortunately users often don't correctly mark reviews as spoilers, making the labels unreliable and limiting the effectiveness of the spoiler alert system based on tags.

The goal of our project is to find an NLP model that can efficiently classify movie reviews as spoilers or non-spoilers using IMDb dataset with a supervised binary classification approach.

2 Related works

To support our project, we begin by examining the available literature, starting with the paper by Rishabh Misra [6], the author of the dataset we've selected. Misra's paper introduces the dataset and provides several insights: paper details, dataset creation, including the tools and methods used and provides code snippets for reading the data and some preliminary lightweight analysis. One of his statistical investigations reveals that users often fail to flag their reviews as containing spoilers, even when they announce spoilers directly within the text. This suggests the need for an NLP solution capable of automatically classifying user reviews as spoiler or non-spoiler.

Misra's also asks himself whether spoiler reviews contain specific movie-related words, such as "Vader" in "Star Wars: The Empire Strikes Back", and finds a correlation: the presence of such words increases the likelihood of a review being a spoiler. The paper also discusses potential expansions, including addressing the movie recommendation problem and genre prediction.

From Chiv's work [4] we take the idea of calculating the similarity between movie reviews and their corresponding synopses to determine whether a reviews is spoiler free. Chiv ignores the "is_spoiler" label to maintain an unsupervised approach, aiming to calculate the cosine similarity between movie synopses and reviews, considering reviews with high similarity scores as spoilers. The results show that reviews with high similarity scores were indeed flagged as spoilers on IMDb. Additionally, reviews marked

as containing spoilers had a higher word difficulty score, which may not indicate the use of more complex vocabulary but rather the presence of typos. This grammatical carelessness could reflect a disregard for the viewing experience of others, implying a higher likelihood of including spoilers.

However, the best results are obtained employing graph-based models. To classify the reviews into spoiler and non-spoiler categories, the models are constructed by combining text, user and movie information. The key to the approach is the use of *Graph Neural Networks* (GNNs) to model the complex relationships between reviews and users. GNNs are particularly well-suited for working with data structured in the form of graphs, such as users social networks and the connections between different elements of reviews and external knowledge[8]. The idea that relying solely on the text of the review is inadequate and that integrating other metadata from the user and the movie is essential for more accurate spoiler detection is supported by various studies:

- Chang et al. [1] encode review sentences and movie genres together to detect spoilers.
- Wan et al. [2] incorporate a *Hierarchical Attention Network*, introducing user bias and item bias.
- Chang et al. [5] exploit syntax-aware GNN to model dependency relations in context words.
- Wang et al. [7] take into account external movie knowledge and user interactions to promote effective spoiler detection.

3 Dataset and metrics used

The dataset used in this project derives from *Kaggle* and is composed of two files: the first one includes information about the reviews, while the second contains the reviewed movies. The whole dataset concerns 1 572 movies, and 573 913 reviews of which 150 924 are spoiler reviews.

3.1 Review dataset

The review dataset includes the following features for each review:

- **review_date**: when the review was posted on IMDb
- **movie_id**: unique ID of the movies, used as a foreign key to link reviews to the corresponding movie information
- **user_id**: unique ID of the user who wrote the review
- **is_spoiler**: binary value indicating whether the review is a spoiler or not (1 for spoiler, 0 for non-spoiler); this is the target variable that our models aim to predict
- **review_text**: text of the review written by the user

- **rating**: movie rating assigned by the user
- **review_summary**: summary of the review text

It's important to note the **class imbalance** in the dataset: 26.3% of the reviews are flagged as spoilers, while 73.7% are flagged as non-spoilers (see Fig. 1 and 2). This imbalance will be addressed during model training to ensure the models perform well in identifying both types of reviews.

Additional statistics:

- Average length of reviews: 1 460 characters
- Minimum length of reviews: 18 characters
- Maximum length of reviews: 14 963 characters

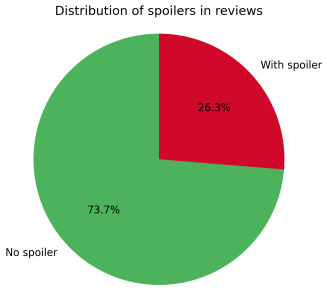

Figure 1: Piechart of spoiler distribution in the reviews dataset

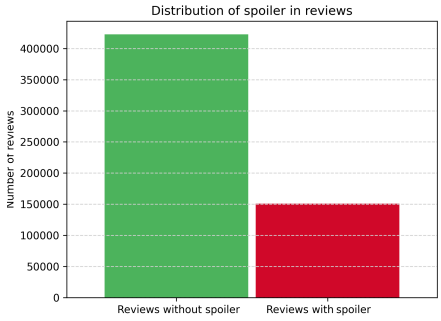

Figure 2: Histogram of spoiler distribution in the reviews dataset

3.2 Movie dataset

- **movie_id**: unique ID of the movies
- **plot_summary**: the movie plot summary
- **duration**: the movie duration
- **genre**: the movie genre
- **rating**: how the movie is rated by the user
- **release_date**: the movie release date
- **plot_synopsis**: the movie plot synopsis

According to Misra's work [6], the plot synopsis appears to be the only informative section for detecting spoilers, while the plot summary proves inadequate due to its lack of pertinent storyline details.

Additional statistics:

- Average length of synopses: 9 644 characters
- Minimum length of synopses: 45 characters
- Maximum length of synopses: 63 904 characters

3.3 Metrics

The results of the various models are evaluated using *accuracy*, *precision*, *recall* and *F1-score*. The focus is on the positive class (spoilers) because accurately detecting spoilers is more critical than identifying non-spoilers. Therefore, the metrics used are chosen to emphasize the importance of the positive class.

4 System overview

For all models, where necessary, we employ an 80/20 training/test set split, always using the same random seed to ensure experiment reproducibility and comparability. Moreover, we set the parameter *stratify* to maintain the same class proportion both in training and test sets.

To establish a foundational benchmark, we start with two simple models: **logistic regression** (see Table 2) and **Naive Bayes** (see Table 3) using 5-fold cross-validation. Both models are applied to the review text (which was preprocessed beforehand) using two different text representation techniques: firstly *bag-of-words* and then *TF-IDF* (Term Frequency-Inverse Document Frequency).

Afterwards, we use three more complex models: **BERT** in the *bert-base-cased* variation, **RoBERTa** in the *roberta-base* (see Table 4) and **RNN** (see Table 5). The RNN architecture includes a *TextVectorization layer* for preprocessing text, an *embedding layer* for dense vector conversion of tokens, a *bidirectional LSTM layer* (256 units) for contextual understanding, and two *dense layer*: a 256-unit *ReLU-activated layer* for feature extraction and a *single-unit output layer*.

Finally, we experiment with some models that leverage also the text of the synopsis, namely **Llama3** in zero-shot mode and **Sentence-BERT.**

4.1 Models details comparison

To obtain better results some models are given as input the preprocessed review text (see section 5.1), while others are given the original review without any re-elaboration (see Table 1). Model parameters are initially selected based on available research papers and Hugging Face models [9] [12], and then refined during the implementation phase through experiments to achieve the best possible performance.

Model	Input text
Logistic regression	preprocessed
Naive Bayes	preprocessed
BERT	original
	preprocessed
RoBERTa	original
	preprocessed
RNN	preprocessed
Sentence Similarity	original (subset)
Llama3	original (subset)

Table 1: Types of input review text for each model

Logistic regression

Bag-of-words

Parameters	Description
C	0.01
class_weight	None
penalty	12
iteration	1500

TF-IDF

Parameters	Description
C	1
class_weight	None
penalty	12
iteration	1500

Table 2: Parameters for logistic regression used with bag-of-words and with TF-IDF

Naive Bayes

Bag-of-words

Parameter	Description
alpha	0.1
class_prior	True

TF-IDF

Parameter	Description
alpha	0.1
class_prior	True

Table 3: Parameters of Naive Bayes used with bag-of-words and with TF-IDF

BERT

Parameter	Value
Model name	bert-base-cased
Batch size	16
Learning rate	2×10^{-5}
Number of epochs	5
Optimizer	AdamW
Weight decay	0.01

RoBERTa

Parameter	Value
Model name	roberta-base
Batch size	16
Learning rate	2×10^{-5}
Number of epochs	5
Optimizer	AdamW
Weight decay	0.01

Table 4: Parameters for BERT and RoBERTa

Recurrent Neural Network

Component	N. of units
TextVectorization layer	-
Embedding layer	256
Bidirectional LSTM layer	256
First Dense layer	256
Outup Dense layer	1

Parameter	Value
Learning rate	1e-4
Weight decay	0.02
Algorithm	Adam
Number of epochs	8

Table 5: Number of units and parameters of RNN

5 Experiments

5.1 Preprocessing

Our analysis involves preprocessing both the review text and the movie plot synopsis, using the NLTK (Natural Language Toolkit) library. Tokenization is performed using the *WordPunctTokenizer* class, which treats individual words and punctuation marks as separate tokens. We leverage NLTK's stopword removal functionality to delete frequently occurring, uninformative words (e.g., "the," "a," "an"). Subsequently, we employ regular expressions to remove any remaining punctuation from the tokenized text. To standardize the morphological variations of words, we employ lemmatization with the *WordNetLemmatizer* from NLTK. Additionally, we apply stemming to compare the effectiveness of both methods, and ultimately choose lemmatization for its superior preservation of word structure.

5.2 Baseline

In order to set a baseline performance, we start our experiments with two simple models: **logistic regression** and **Naive Bayes** (Multinomial variant). To apply them, we create two text representations for each model: **TF-IDF** and **bag-of-words**. We divide the dataset into training and test sets. We make grid search with 5-fold cross-validation to find the best parameters of each model involving the two text representations. In particular, for logistic regression we fix a number of iterations and use grid search to find the regularization parameters and class weight parameter that indicates whether to use different weights for classes during model training. Meanwhile, for Naive Bayes, we use grid search to find the smoothing parameter and the fit prior parameter which determine whether to estimate prior probabilities or not. Once the parameters are defined, we train the model using k-fold cross-validation with K=5. Then, we test the models on the test set (see Section 6.1)

5.3 BERT and RoBERTa

We fine-tune two transformer models **BERT** and **RoBERTa** using the base cased model version. For both models, we used the original review text as well as the preprocessed text. We divide the dataset into three parts: a development set (80%) and a test set (20%), then we divide again the development set into a **training set** (80%)

and an **evaluation set** (20%). We apply tokenization to both sets, setting the sequence maximum length to 512 tokens; therefore sequences shorter than 512 tokens are padded with extra tokens to reach this length, while sequences longer than 512 tokens are truncated. We transform the pandas dataset into a Hugging Face dataset, and then we define the training parameters using the *TrainingArguments* class for training the models (see Section 6.2).

5.4 Recurrent Neural Network

For RNN we use the preprocessed reviews, splitting the dataset into train and test set and then converting it into a TensorFlow dataset. Before training the network, we randomly shuffle the dataset, in order to improve the convergence of training. We group the examples of training and test set into batches with size 64. To transform the reviews into a representation usable by the network, we use TensorFlow's *TextVectorization* to create an embedding of the reviews. The number of epochs and the learning rate are chosen arbitrarily by conducting several trials and changing the parameters (see Table 14).

5.5 Sentence similarity

To enhance our analysis beyond movie reviews alone, we incorporate the corresponding movie synopsis, assuming that the similarity between these elements could provide valuable insights. To achieve this, we employ spaCy's *sentence_bert* integration to calculate the semantic similarity between reviews and their synopses. This approach combines the natural language processing capabilities of spaCy with the high-quality sentence embeddings provided by Sentence-BERT (SBERT). The idea is to calculate the cosine similarity and define a threshold to decide whether a comment is a spoiler or not. Instead of comparing the entire movie synopsis, we focus only on the last 512 characters, because this approach is more likely to capture important plot events typically found towards the end of synopses [3].

We first test the algorithm on a dataset subset (250 000 rows): we obtain values of similarity which indicates that it is not possible to define a threshold since it doesn't seem to be any correlation between the similarity's value and the *is_spoiler* label. We decide to not proceed further, as presumably we would obtain the same result on the entire dataset.

5.6 Zero-shot classification with Llama

Another approach to integrate the plot into the classification is to include it in the prompt along with the review. In this case, we choose to use **Llama3**.

Due to the high computational cost in terms of time, we extract a subset of 10 000 reviews and we merge them with the movie plot using *movie_id* as the foreign key. We apply Llama using the whole text of the movie plot, but after seeing the high time complexity and the poor results we decide to limit the movie plot only to the last 512 characters. This approach reduces the execution time and leads to an improvement of the results (see Table 16).

6 Results and Analysis

During the analysis of the obtained results for each model, it is important to put more focus on the recall metric, rather than on the accuracy, since the dataset is imbalanced towards the negative class and the results could be biased by this.

6.1 Results on baseline

6.1.1 Logistic regression results

The low metric scores obtained with the logistic regression both in the training and in the test set (see Tables 6 and 7), suggest the need of more complex models to achieve better results.

TF-IDF

Dataset	Accuracy	Recall	Precision	F1-score
Training set	0.78	0.33	0.66	0.44
Test set	0.78	0.33	0.66	0.45

Table 6: Result for logistic regression with TF-IDF

Bag-of-words

Dataset	Accuracy	Recall	Precision	F1-score
Training set	0.77	0.29	0.67	0.41
Test set	0.78	0.30	0.67	0.41

Table 7: Results for logistic regression with bag-of-words

6.1.2 Naive Bayes results

Similarly to the logistic regression, also Naive Bayes doesn't achieve good results, particularly experiencing a worsening of the recall using the TF-IDF text representation (see Tables 8 and 9.

TF-IDF

Dataset	Accuracy	Recall	Precision	F1-score
Training set	0.77	0.24	0.65	0.34
Test set	0.75	0.13	0.66	0.22

Table 8: Results for Naive Bayes with TF-IDF

Bag-of-words

Dataset	Accuracy	Recall	Precison	F1-score
Training Set	0.73	0.46	0.50	0.47
Test Set	0.76	0.33	0.56	0.41

Table 9: Results for Naive Bayes with bag-of-words

6.2 BERT and RoBERTa results

BERT and RoBERTa achieve the best results across all metrics, particularly the recall with RoBERTa (see Tables 10 and 12). However, when testing the models on the preprocessed dataset, the results for both models worsen (see Tables 11 and 13). Indeed, removing both stopwords and punctuation can alter the context of sentences and potentially lead to misinterpretations by the model.

Original dataset

Dataset	Accuracy	Recall	Precision	F1-score	Time
Training set	0.78	0.48	0.61	0.54	$\sim$ 5h 30m
Test set	0.78	0.49	0.61	0.54	

Table 10: Results for BERT on the original dataset without preprocessing

Preprocessed dataset

Dataset	Accuracy	Recall	Precision	F1-score	Time
Training set	0.78	0.48	0.59	0.53	$\sim$ 5h
Test set	0.78	0.48	0.60	0.53	

Table 11: Results for BERT on the preprocessed dataset

Original dataset

Dataset	Accuracy	Recall	Precision	F1-score	Time
Training set	0.80	0.53	0.64	0.58	$\sim$ 5h 50m
Test set	0.80	0.54	0.64	0.58	

Table 12: Results for RoBERTa on the original dataset without preprocessing

Preprocessed dataset

Dataset	Accuracy	Recall	Precision	F1-score	Time
Training set	0.79	0.48	0.63	0.55	$\sim$ 5h 30m
Test set	0.79	0.49	0.63	0.55	

Table 13: Results for RoBERTa on the preprocessed dataset

6.3 RNN results

With the recurrent neural network, we achieve similar results to the fine-tuning done using BERT and RoBERTa (see Tables 14 and Section 6.2) however, we still cannot achieve good results on the recall.

Dataset	Accuracy	Recall	Precision	F1-score	Time
Training set	0.82	0.41	0.79	0.54	$\sim$ 40m
Test set	0.78	0.34	0.64	0.44	

Table 14: Results for RNN

6.4 Sentence similarity results

As can be seen from Table 15, the accuracy and recall results are similar to 0.50. This is because the average cosine similarity value of the spoiler and non-spoiler classes is very close to each other, making it difficult to define a threshold for similarity to assign the label.

Dataset	Accuracy	Recall	Precision	F1-score	Time
Subset	0.54	0.56	0.30	0.39	$\sim$ 3h 30m

Table 15: Results for Sentence Similarity

6.5 Llama3 results

Even in this case, we encountered two main issues using the movie plot as part of the zero-shot learning prompt. Firstly, the computation is so burdensome that the model takes a long time even on a few lines. Additionally, despite trying various prompts, we don't achieve good results on the 10 000 rows subset used (see Section 5.6 and Table 16). The recall value is higher in this model because it flags almost all comments as spoilers.

Dataset	Accuracy	Recall	Precision	F1-score	Time
Subset	0.36	0.82	0.27	0.40	$\sim$ 3h 30m

Table 16: Results for zero-shot classification

7 Conclusions

The objective of our work is to find a model which can efficiently identify reviews containing spoilers, to help users on the web discerning which content to read before watching a movie.

We are looking for a trade-off among all the metrics putting more emphasis on the recall measure, since we want to make less mistakes on the classification of the positive class (spoilers).

From the experiments conducted the model which performs better on our dataset, both on training and test set, is RoBERTa applied on the original text of the reviews (see Table 12) even though the recall has not reached an optimal value.

Exploiting the plot of the movie does not help to obtain better results (see Sections 5.5 and 6.5). However, we observe that extracting the last part of the plot is more effective than using the whole text.

As the literature suggests, one promising direction for future works is the integration, not only of the movie plot, but also of the metadata related to the reviews' authors, for example the tendency of the user of making spoilers. Furthermore, the implementation could be improved by performing grid search for BERT, RoBERTa and RNN. Naturally, this approach would require more resources due to the large dataset size.

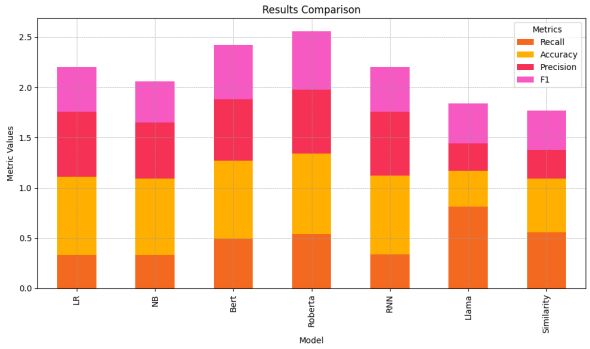

Figure 3: Comparison of best results for each model on test set

8 Instructions to execute and list of files

8.1 List of files

There are several notebooks. In the notebook *Data_Exploration_Preprocessing.ipynb*, we explore the dataset and include various functions for the preprocessing. In the file *Baseline.ipynb* we have simpler models used as baseline, such as Naive Bayes and logistic regression. The files *Fine_tuneBERT.ipynb* and *Fine_tuneRoBERTa.ipynb* are the notebooks where we do the fine-tuning of BERT and RoBERTa. In the file *RNN.ipynb*, there is the implementation of the RNN, while in the *Llama3.ipynb* file, we find the zero-shot classification done with Llama. Finally, we have the file *Sentence_Similarity.ipynb*: here is the implementation that computes the cosine similarity between the review text and the plot synopsis of the movie.

8.2 Instructions to execute

- Create a virtual enviroment: `python3 -m venv name`
- Activate the enviroment: `source name/bin/activate`
- Install the libraries: `pip install -r requirements.txt`
- Install the torch library with CUDA support: `pip3 install torch torchvision torchaudio -index-url https://download.pytorch.org/whl/cu121` [10]
- Install tensorflow with CUDA support [11]:
  - Linux: `pip install tensorflow[and-cuda]`
  - Windows: `pip install tensorflow`, TensorFlow 2.10 was the last version of TensorFlow to support GPU on native Windows.
- Run Jupyter Notebook: `jupyter lab`

References

[1] Buru Chang et al. "A Deep Neural Spoiler Detection Model Using a Genre-Aware Attention Mechanism". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 2018. URL: https://api.semanticscholar.org/CorpusID:49313814.

[2] Mengting Wan et al. "Fine-Grained Spoiler Detection from Large-Scale Review Corpora". In: *CoRR* abs/1905.13416 (2019). arXiv: 1905.13416.

[3] Sreejita Biswas and Goutam Chakraborty. *MOVIE REVIEWS: TO READ OR NOT TO READ! Spoiler Detection with Applied Machine Learning*. https://support.sas.com/resources/papers/proceedings20/5012-2020.pdf. 2020.

[4] Kevin Chiv. *IMDb Review Spoiler Detection*. https://github.com/kevinchiv/IMDb-Review-Spoiler-Detection. 2020.

[5] Buru Chang et al. *"Killing Me" Is Not a Spoiler: Spoiler Detection Model using Graph Neural Networks with Dependency Relation-Aware Attention Mechanism*. 2021. arXiv: 2101.05972.

[6] Rishabh Misra. *IMDB Spoiler Dataset*. Sept. 2022. arXiv: 2212.06034.

[7] Heng Wang et al. *Detecting Spoilers in Movie Reviews with External Movie Knowledge and User Networks*. 2023. arXiv: 2304.11411.

[8] Zinan Zeng et al. *MMoE: Robust Spoiler Detection with Multi-modal Information and Domain-aware Mixture-of-Experts*. 2024. arXiv: 2403.05265.

[9] bhavyagiri. *roberta-base-finetuned-imdb-spoilers*. https://huggingface.co/bhavyagiri/roberta-base-finetuned-imdb-spoilers.

[10] *PyTorch documentation*. https://pytorch.org/get-started/locally/. Accessed: June 24th, 2024.

[11] *Tensorflow documentation*. https://www.tensorflow.org/install/pip?hl=it. Accessed: June 24th, 2024.

[12] Zritze. *imdb-spoiler-bertOrigDataset*. https://huggingface.co/Zritze/imdb-spoiler-bertOrigDataset.