Spoiler detection and extraction pt. 2
POC for NLP Course, Winter 2022/2023

M. Kierznowski, Ł. Pancer, P. Wesołowski
Warsaw University of Technology

supervisor: Anna Wróblewska
Warsaw University of Technology
anna.wroblewska1@pw.edu.pl

Abstract

This document presents exploratory data analysis, describes current experiments, and provides preliminary results for the second NLP course project. The project addresses the spoiler detection task. The main goal is to develop and evaluate architectures that are able to detect spoiler phrases within sentences. So far, we have focused on creating deep learning models based on LSTM networks.

1 Introduction

As an introduction, let us remind that in the previous project, we classified entire reviews as either containing spoilers or not. Then, we employed explainability tools to find the phrases that determined the classification of the entire review as a spoiler. In this project, we extend the previous work. Instead of teaching the models to classify entire reviews, we want to train them to extract spoiler phrases at the word level. Such a task may be viewed as a classification task since we aim to analyze models which classify every single word from a given input either as a spoiler or non-spoiler.

Since the problem posed in this way requires an accurately annotated dataset, we only have the option of using one of the three used in the previous project. We transform this dataset appropriately to fit our task. Finally, we perform some preliminary experiments with the LSTM-based deep learning model.

Section 2 contains brief literature and currently employed LSTM model review. Section 3 contains exploratory data analysis and transformation of the dataset. Finally, our preliminary experiment is described in section 4.

2 Literature

2.1 Phrase extraction

Our project idea was inspired by a medium article (Dash, 2021), an overview of a solution to a Twitter sentiment extraction Kaggle competition(Maggie, 2020). The competition brought up the subject of extracting the exact part of a sentence responsible for its sentiment. For example, in a tweet, ”my boss is bullying me,” the phrase ”bullying me” is directly responsible for its negative sentiment, as the rest of the sentence is neutral and could be used in a sentence of any sentiment. Unfortunately, this exact problem is, to our knowledge, non-existent in current literature. However, a lot can be discovered from problems of similar nature.

Zhang (2011) proposed a method for selecting words that do not bear sentiment on their own but only given context. For example, words describing resources may be positive or negative, e.g., ”The washer uses a lot of electricity” or ”The washer uses little water.” In (Madasu and Elango, 2020), authors study the impact of a number of machine learning classifiers on the Term Frequency Inverse Document Frequency (TF-IDF) technique. Using feature selection combined techniques, they were able to outperform neural networks in sentiment analysis. Jianqiang et al. in (2018) applied a word embeddings method to sentiment analysis of Twitter posts. It’s an improvement over TF-IDF because it extracts features such as contextual semantic relationships and co-occurrence statistical characteristics between words. Their approach, based on deep convolutional neural networks, outperformed the baseline word n-gram models. Yet another work (Rezaeinia et al., 2017) introduces an Improved Word Vectors (IWV) method, improving the accuracy of pretrained word embeddings.

2.2 Long short-term memory

Long short-term memory (LSTM) architecture is an altered version of the recurrent neural networks (RNNs) that is able to learn long-term dependencies, mostly in sequence prediction problems. This type of network is used in speech recognition, machine translation, forecasting, etc.

Originally LSTM architecture was introduced in the work of Hochreiter, and Schmidhuber (1997). The main goal of LSTM is to mitigate the long-term dependency problem in comparison to the RNNs.

The main structure of the LSTM network is depicted in figure 1

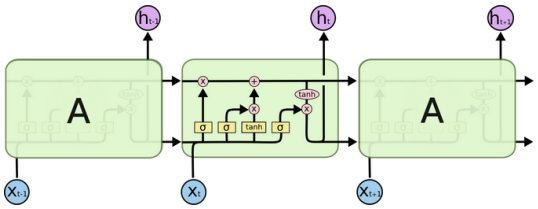

Figure 1: An example of LSTM module

All LSTM networks have the form of a chain of repetitive modules. Contrary to an ordinary RNN block, LSTM consists of three different gates: a forget gate, an input gate, and an output gate. Those gates are crucial for obtaining long short-term memory in LSTM architecture.

The first major challenge the LSTM tackles is what information consider redundant. For that issue, the forget gate comes to play. Example of forget gate is illustrated in figure 2

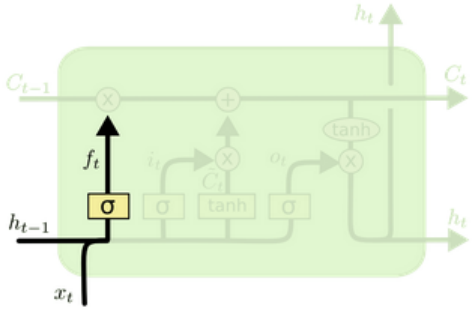

Figure 2: Forget gate

The disregard of information is done by the sigmoid layer. Whereas 1 represents "completely keep this," a 0 represents "completely get rid of this." Afterward, the redundant information from the input and previous hidden state is discarded.

Subsequently, the LSTM block uses the input gate, presented in figure 3, to determine what new information is going to be stored in the cell state. This has two parts. First, a sigmoid layer decides which values are being updated. Next, a tanh layer creates a vector of new candidate values that could be added to the state. In the final step, the LSTM cell combines them to create an update to the state.

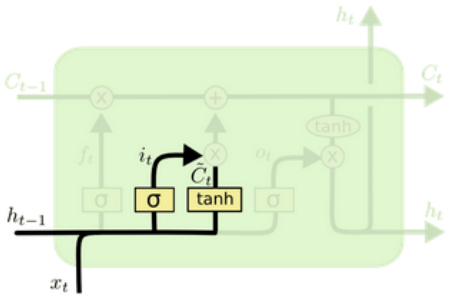

Figure 3: Input gate

Finally, LSTM uses a sigmoid layer that decides which parts of the input and hidden state will be considered as an output, which is shown in figure 4. In addition, the tanh layer is utilized to map the cell state values into $[-1, 1]$ interval and multiply them by the output of the sigmoid.

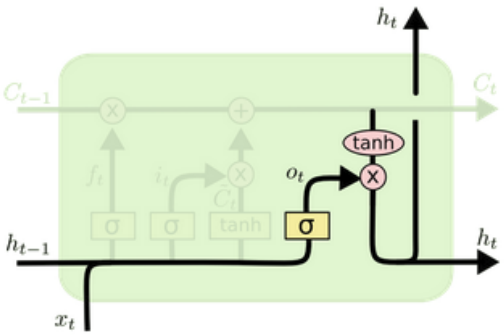

Figure 4: Output gate

Those improvements to the vanilla RNN's architectures resolve problems with long-term memory making the LSTM architectures superior in many tasks.

3 Dataset

In our work, we needed a dataset that features word-level annotations. This need followed from the definition of the task. To our knowledge, only one dataset provides such annotations, namely the

TV Tropes Books dataset[1]. The dataset consists of precisely annotated reviews. In total, it contains 340k reviews with a 1:4 spoiler-to-non-spoiler review ratio. Among these, 670k sentences and 110k spoiler sentences are present. The dataset has already been split into the training, validation, and testing subsets. Note that in our architectures, we need text samples with a specified maximum number of words (tokens). That's why we modified this dataset to fit our needs. First, we present the exploratory data analysis, and then we describe obtaining the final dataset, adjusted to our task, that is used in the actual experiments.

3.1 Exploratory data analysis

The dataset contains serialized JSON objects. An example object corresponding to a single review is shown in the listing 1. Recall that our final model has a fixed input tokens count and the same size output vector. Therefore, our exploratory data analysis focuses on choosing the right strategy to transform this dataset into a form that meets our expectations. The following exploratory data analysis is based on the training part of the dataset.

Listing 1: Sample truncated JSON object in the TV Tropes Books dataset. Each sentence contains a boolean flag that identifies spoiler sentences. In addition, annotated character indices provide specific spoiler boundaries.

```
{
'page': 'https://tvtropes.org/...',
'trope': 'Kill the Cutie',
'has_spoiler': True,
'sentences': [[True, 'Walter, who was
    the ...', [[0, 89]]]]
}
```

First, we decided to verify how many spoilers are annotated in a single sentence. Figure 5 shows the corresponding histogram.

Then, due to the expected fixed input word count, we focused on the histograms of the word count. Figure 6 shows the histogram of word counts in the whole spoiler reviews. Another perspective is provided in figure 7 depicting the histogram of word counts in sentences (only the sentences in spoiler reviews are considered).

As we were looking for an idea of how to transform the collection appropriately to get review fragments with a certain maximum word count, we decided to check where in the sentences the

[1]available at https://github.com/rzepinskip/tvtropes-books

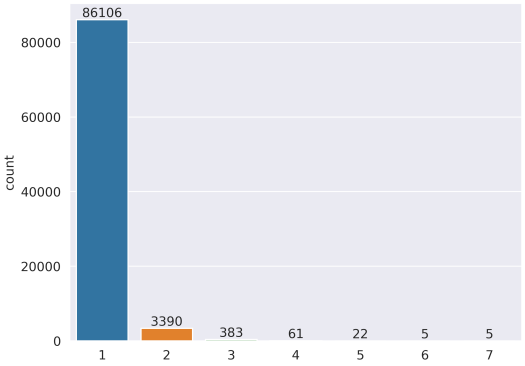

Figure 5: Histogram presenting the distribution of spoiler counts within sentences. Note that only sentences with at least one spoiler are included. Sentences containing only one spoiler dominate.

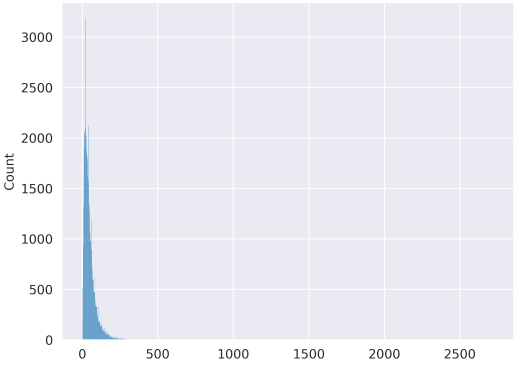

Figure 6: Histogram presenting the distribution of word counts in whole spoiler reviews.

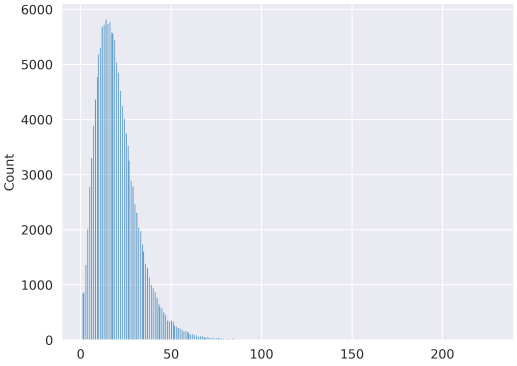

Figure 7: Histogram presenting the distribution of word counts in sentences. The sentences come from spoiler reviews.

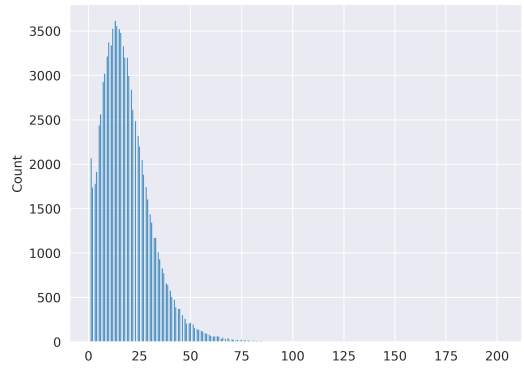

Figure 8: Histogram presenting the distribution of the last spoiler word inside a spoiler sentence.

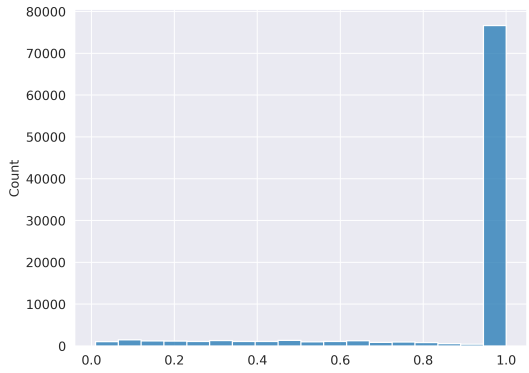

Figure 9: Histogram presenting the distribution of the fraction of spoiler words in spoiler sentences.

spoilers end up. The result is shown in figure 8.

The similarity of figures 7 and 8 seemed suspicious to us. Therefore, we decided to verify what fractions of spoiler sentences are actually annotated as spoilers. For every spoiler sentence, the number of words annotated as spoilers is divided by the number of words in the sentence. The histogram shown in figure 9 shows the distribution of these fractions.

The plot presented in figure 9 may be surprising. It turns out that in the vast majority, entire sentences are spoilers. We do not know whether this is due to fairly general tagging by portal users who have tagged whole sentences or whether whole sentences are actually spoilers.

3.2 Dataset transformation

The above exploratory data analysis led us to decide to finally transform the dataset in a straightforward manner. Recall that we had to make sure that our input samples were not longer than the specified number of tokens. The algorithm is as follows. For every spoiler review, perform the following operations:

1. Concatenate sentences as long as they are within the maximum number of words.
2. Add concatenated sentences to the final dataset.
3. If some sentences from the review considered have not been included in the result, go back to 1. and start concatenating them.

Of course, single sentences that are longer on their own than the maximum number of words allowed are not included in the resulting dataset. It is worth noting that this transformation must be combined with procedures typically performed by a tokenizer, such as splitting. It's due to the need to track the positions of spoiler words. Therefore, simultaneously with the concatenation operations, the target label vector is created. The length of the vector equals the number of words in the corresponding review fragment. Such a vector contains boolean flags indicating spoiler words. Example data and corresponding target vector are presented in listing 2.

4 Preliminary architecture

Since we were afraid of the computational complexity, we selected the maximum length of review fragments to be 64 tokens. We ended up with about 47k review fragments that contained both spoiler and non-spoiler phrases, 20k fragments containing only spoiler words, and 12k fragments with no spoilers. Therefore, our final training dataset features about 79k labeled review fragments.

Before describing specific layers of the model, some key data transformation has to be prepared. For samples that contain less than 64 words, we applied zero padding such that every document length is the same, equal to 64. After those necessary prepossessing, architecture is able to consume the data.

The architecture consists of an LSTM layer which is preceded by an embeddings layer. This specific object transfers tokenized input to its embedded value. As weights, we utilize GloVe embeddings, and those weights are frozen (they are

Listing 2: Sample element of the dataset adjusted to our needs. The maximum number of tokens was 64. Note that the text has already undergone preprocessing, e.g., punctuation removal. The target vector is a boolean vector indicating spoiler words. In this case, *and can't bear him children* was annotated as a spoiler phrase.

```
("celia is this to johnny because he's from a rich well to do family while she's
white trash from sugar ditch and doesn't know how to cook clean and can't bear him
children",
 [False, False, ..., False, True, True, True, True, True])
```

no longer trainable). Since our model aims to classify every word, it might be considered a sequence-to-sequence model. Therefore, the number of neurons in the last layer corresponds to the maximum number of input words (in provided case, 64). This first approach, combined with independently applied sigmoid function, gave us satisfactory results. On the testing set, we achieved 0.8169 accuracy, where accuracy is considered per each word. Apart from that, we also implemented the Jaccard similarity score, which gave us a 0.7811 set similarity for non-spoiler words and 0.4718 set similarity for spoiler words. The exact hyperparameters, etc., will be provided in the final report because we may change the model, etc.

5 Summary

In this document, we have shown our current progress. The next step is to increase the input and output vector size (from 64 to, probably, 512 - recall figure 6). In addition, we will test different models.

References

Jitendra Dash. 2021. Extract the right phrase from sentence. `https://medium.com/analytics-vidhya/extract-the-right-phrase-from-sentence-29aa5f8b9182`.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhao Jianqiang, Gui Xiaolin, and Zhang Xuejun. 2018. Deep convolution neural networks for twitter sentiment analysis. *IEEE access*, 6:23253–23260.

Avinash Madasu and Sivasankar Elango. 2020. Efficient feature selection techniques for sentiment analysis. *Multimedia Tools and Applications*, 79(9):6313–6335.

Wei Chen Maggie, Phil Culliton. 2020. Tweet sentiment extraction. `https://kaggle.com/competitions/tweet-sentiment-extraction`.

Seyed Mahdi Rezaeinia, Ali Ghodsi, and Rouhollah Rahmani. 2017. Improving the accuracy of pre-trained word embeddings for sentiment analysis. *arXiv preprint arXiv:1711.08609*.

Lei Zhang and Bing Liu. 2011. Extracting resource terms for sentiment analysis. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1171–1179.